Object Storage Service

C SDK Developer Guide

Issue 01

Date 2025-02-07

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

This section describes the version compatibility and important notes about the C SDK of Object Storage Service (OBS).

Compatibility

- **3.*.*** is compatible with **3.0.0**.
- **3.*.*** is incompatible with **2.*.***.
- **3.*.*** is incompatible with **1.*.***.
- Arm compilation environment:

```
NAME="EulerOS"
VERSION="2.0 (SP8)"
ID="euleros"
ID_LIKE="rhel fedora centos"
VERSION_ID="2.0"
PRETTY_NAME="EulerOS 2.0 (SP8)"
ANSI_COLOR="0;31"
```

  Kernel version:

```
4.19.36-vhulk1905.1.0.h276.eulerosv2r8.aarch64
```

  gcc/g++ version:

```
gcc (GCC) 10.3.0/g++ (GCC) 10.3.0
```

- Linux compilation environment:

```
NAME="CentOS Linux"
VERSION="7 (Core)"
ID="centos"
ID_LIKE="rhel fedora"
VERSION_ID="7"
PRETTY_NAME="CentOS Linux 7 (Core)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:centos:centos:7"
HOME_URL="https://www.centos.org/"
BUG_REPORT_URL="https://bugs.centos.org/"

CENTOS_MANTISBT_PROJECT="CentOS-7"
CENTOS_MANTISBT_PROJECT_VERSION="7"
REDHAT_SUPPORT_PRODUCT="centos"
REDHAT_SUPPORT_PRODUCT_VERSION="7"
```

  Kernel version:

```
3.10.0-957.5.1.el7.x86_64
```

  gcc/g++ version:

```
gcc (GCC) 10.3.0/g++ (GCC) 10.3.0
```


NOTE

The required environments for compiling the SDK binary package are listed above. Compatibility is not guaranteed for other kernels and operating systems. If you need to use other operating systems or kernel versions, use the open-source code to compile on your own.

Important Notes

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

- Ensure that you are familiar with OBS basic concepts, such as **buckets**, **objects**, and **access keys (AK and SK)**.
- After an API call is complete using an instance of ObsClient, view whether an exception is thrown. If no, the API call was successful. If yes, the operation failed. You can check the error information from **SDK Error Handling**.
- Some features are available only for some regions. If **405** HTTP status code is returned for a certain feature API, check whether the region supports that feature.

2 Downloading and Installing the SDK

This section provides the download and compilation methods for the OBS C SDK.

SDK Download

- Latest version of OBS C SDK source code: Click **here** to download.

SDK Compilation

You can compile the SDK based on the platform you are using. Before that, you must obtain the **SDK source code**.

- In Linux:

Go to the **source/eSDK_OBS_API/eSDK_OBS_API_C++/** directory and run the following script:

export SPDLOG_VERSION=spdlog-1.9.2

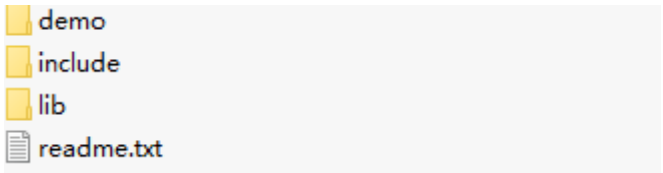
#On an x86 server, run the following command:

bash build.sh sdk

#On an Arm server, run the following command:

bash build_aarch.sh sdk

For details about the parameters, see the comments in the scripts. A **sdk.tgz** demo package that contains the content shown below is generated.



- In Windows:

Use Visual Studio to open the **sln** file in **source**/**eSDK_OBS_API**/**eSDK_OBS_API_C++**/**sln**/vc100/ and generate the **obs** project. Then, **huaweisecurec.lib**, **huaweisecurec.dll**, **libeSDKOBS.lib**, and **libeSDKOBS.dll** are generated in the output directory (which can be queried in the project properties).

3 Quick Start

3.1 Setting Up an OBS Environment

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

Step 1 Sign up for a cloud service account.

Create an account to use OBS. If you already have one, use it instead.

1. Open a browser.
2. Visit the **Huawei Cloud official website**.
3. In the upper right corner of the page, click **Register**.
4. Enter the registration information and click **Register**.

Step 2 Subscribe to OBS.

You are required to top up your account before using OBS.

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

Step 3 Create access keys.

OBS uses the AK and SK of an account for signature verification to make sure that only authorized accounts can access specified OBS resources. Detailed explanations about AK and SK are as follows:

- An access key ID (AK) defines a user who accesses the OBS system. An AK belongs to only one user, but one user can have multiple AKs. The OBS system recognizes the users who access the system by their access key IDs.
- An SK is the secret access key on OBS, which is required 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.

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

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

     NOTE

     - If you have not bound an email address or mobile number, you need to enter only the login password.
     - If you have bound an email address and a mobile number, you can select the verification either by email address or mobile number.

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

     NOTE

     - Each user can create up to two valid access keys.
     - To prevent your access keys from being leaked, keep them secure. If you click **Cancel** in the dialog box, the access key will not be downloaded, and can neither be obtained later. If you need to use an access key later, you need to create a new one.
- 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.

To obtain them, see **Obtaining a Temporary AK/SK and a Security Token**.

NOTICE

OBS is a global service. When obtaining temporary access keys, set the token scope to **domain** to apply the token to global services. Global services are not differentiated by any project or region.

----End

3.2 Service Address

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

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

3.3 Initializing the SDK

ObsClient is a client used to access OBS in C language. It offers a series of APIs for interacting with OBS. You can use these APIs to manage and operate resources (such as buckets and objects) stored in OBS.

Before using the OBS C SDK to initiate an OBS request, you need to call the initialization API. When you are exiting the process, you need to call the initialization cancellation API to release resources.

Before using the C SDK, you must first call **obs_initialize** to complete the initialization. This API only needs to called once in a process.

```
obs_status  ret_status = OBS_STATUS_BUTT;
ret_status = obs_initialize(OBS_INIT_ALL);
if (OBS_STATUS_OK != ret_status)
{
    printf("obs_initialize failed(%s).\n", obs_get_status_name(ret_status));
    return ret_status;
}
obs_deinitialize();
// Do not repeatedly call obs_initialize or obs_deinitialize, because these operations may lead to invalid memory access.
```

3.4 Initializing option

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

When the function of C SDK is called, **option** must be input. You can use function **init_obs_options** to initialize the **option** configuration, and set AK, SK, endpoint, bucket, timeout period, and temporary authentication by using **option**.

- Sample code for creating and initializing **option** using permanent access keys (AK/SK):

```
// Create and initialize option.
obs_options option;
init_obs_options(&option);
option.bucket_options.host_name = "<your-endpoint>";
option.bucket_options.bucket_name = "<Your bucketname>";

// 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
// Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
```

- Sample code for creating and initializing **option** using temporary access keys (AK/SK and SecurityToken):

```
// Create and initialize option.
obs_options option;
init_obs_options(&option);
option.bucket_options.host_name = "<your-endpoint>";
option.bucket_options.bucket_name = "<Your bucketname>";

// 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
// Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
option.bucket_options.token = getenv("SecurityToken");
```

  NOTE

  - OBS is a global service. When obtaining temporary credentials, set the token scope to **domain** to make the token applicable to all projects and regions within an account.

3.5 Creating a Bucket

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

A bucket is a global namespace of OBS and is a data container. It functions as a root directory of a file system and can store objects. Sample code:

```
static void test_create_bucket(obs_canned_acl canned_acl, char *bucket_name)
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
```



```
    option.bucket_options.bucket_name = "<Your bucketname>";
// 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");


    // Set the response callback function.
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };

    // Create a bucket. For details about the pre-defined access policy, see section Managing Bucket ACLs.
    create_bucket(&option, "<bucket ACL>", NULL, &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("create bucket successfully. \n");
    }
    else
    {
        printf("create bucket failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

NOTE

Bucket names are globally unique. Ensure that the bucket you create is named differently from any other bucket. A bucket name must comply with the following rules:

- Contains 3 to 63 characters, starts with a digit or letter, and supports only lowercase letters, digits, hyphens (-), and periods (.)
- Cannot be an IP address.
- Cannot start or end with a hyphen (-) or period (.).
- Cannot contain two consecutive periods (.), for example, **my..bucket**.
- Cannot contain periods (.) and hyphens (-) adjacent to each other, for example, **my-.bucket** or **my.-bucket**.
- If you create buckets of the same name, no error will be reported and the bucket properties comply with those set in the first creation request.

The bucket created in the previous example is of the default ACL (**private**), in the OBS Standard storage class, and in the default location where the global domain resides.

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

NOTICE

- During bucket creation, if the endpoint you use corresponds to the default region CN North-Beijing1 (cn-north-1), specifying a region is not a must. If the endpoint you use corresponds to any other region, except the default one, you must set the region to the one that the used endpoint corresponds to. To view the valid regions, see **Regions and Endpoints**. For example, if the endpoint used for initialization is **obs.ap-southeast-1.myhuaweicloud.com**, you must set **Location** to **ap-southeast-1** when you create a bucket. Otherwise, status code 400 will be returned.

3.6 Uploading an Object

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

The data flow is saved to **callback_data** (see the **callback_data** definition in **Performing a Streaming Upload**). Use the callback function **put_object_data_callback** defined in **obs_put_object_handler** to copy the content of the uploaded object to **buffer**, a character pointer of the callback function parameter. Sample code:

```
static void test_put_object_from_buffer()
{
    // Buffer to be uploaded
    char *buffer = "abcdefg";
    // Length of the buffer to be uploaded
    int buffer_size = strlen(buffer);
    // Name of an object to be uploaded
    char *key = "put_buffer_test";

    // Initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucket name>";
// 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // Initialize the properties of an object to be uploaded.
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    // Customize the structure for storing uploaded data.
    put_buffer_object_callback_data data;
    memset(&data, 0, sizeof(put_buffer_object_callback_data));
    // Assign the buffer value to the structure.
    data.put_buffer = buffer;
    // Set buffersize.
    data.buffer_size = buffer_size;
    // Set the callback function. The corresponding callback function needs to be implemented.
    obs_put_object_handler putobjectHandler =
    {
        { &response_properties_callback, &put_buffer_complete_callback },
          &put_buffer_data_callback
    };
    put_object(&option, key, buffer_size, &put_properties,0,&putobjectHandler,&data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("put object from buffer successfully. \n");
    }
    else
    {
        printf("put object from buffer failed(%s).\n", obs_get_status_name(data.ret_status));
    }
}
```


NOTE

For more information, see **Performing a Streaming Upload**.

3.7 Downloading an Object

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

Sample code:

```
static void test_get_object()
{
    char *file_name = "./test";
    obs_object_info object_info;
    // Initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";
// 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set the object to be downloaded.
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key = "<object key>";
    object_info.version_id = "<object version ID>";
    // Customize the structure for storing downloaded object data based on service requirements.
    get_object_callback_data data;
    data.ret_status = OBS_STATUS_BUTT;
    data.outfile = write_to_file(file_name);
    // Define range download parameters.
    obs_get_conditions getcondition;
    memset(&getcondition, 0, sizeof(obs_get_conditions));
    init_get_properties(&getcondition);
    // Customize callback function for download.
    obs_get_object_handler get_object_handler =
    {
        { &response_properties_callback, &get_object_complete_callback},
        &get_object_data_callback
    };
    get_object(&option, &object_info, &getcondition, 0, &get_object_handler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("get object successfully. \n");
    }
    else
    {
        printf("get object faied(%s).\n", obs_get_status_name(data.ret_status));
    }
    fclose(data.outfile);
}
```

NOTE

For more information, see **Downloading an Object**.

3.8 Listing Object

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

Sample code:

```
static void test_list_bucket_objects()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

// 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // Set response callback function.
    obs_list_objects_handler list_bucket_objects_handler =
    {
        { &response_properties_callback, &list_objects_complete_callback },
        &list_objects_callback
    };

    // Customize callback data.
    list_bucket_callback_data data;
    memset(&data, 0, sizeof(list_bucket_callback_data));
    // List objects.
    list_bucket_objects(&option, "<prefix>", "<marker>", "<delimiter>", "<maxkeys>", &list_bucket_objects_handler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("list bucket objects successfully. \n");
    }
    else
    {
        printf("list bucket objects failed(%s).\n",
            obs_get_status_name(data.ret_status));
    }
}
```

NOTE

- For more information, see **Listing Objects**.

3.9 Deleting an Object

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

Sample code:

```
static void test_delete_object()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize object information.
    obs_object_info object_info;
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key = "<Your Key>";
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

// 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_response_handler resqonseHandler =
    {
        &response_properties_callback,
        &response_complete_callback
    };
    // Delete an object.
    delete_object(&option,&object_info,&resqonseHandler, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {
        printf("delete object successfully. \n");
    }
    else
    {
        printf("delete object failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

NOTE

- For more information, see **Deleting Objects**.

4 Initialization

4.1 Configuring the AK and SK

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

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

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

- **SDK Initialization**
- **Configuring option**
- **Configuring SDK Logging**

4.2 SDK Initialization

NOTICE

If you have any questions during the development, post them on the **Issues** page of GitHub.

ObsClient is a client used to access OBS in C language. It offers a series of APIs for interacting with OBS. You can use these APIs to manage and operate resources (such as buckets and objects) stored in OBS.

Before using the OBS C SDK to initiate an OBS request, you need to call the initialization API. When you are exiting the process, you need to call the initialization cancellation API to release resources.

Before using the C SDK, you must first call **obs_initialize** to complete the initialization. This API only needs to called once in a process.

```
obs_status  ret_status = OBS_STATUS_BUTT;
ret_status = obs_initialize(OBS_INIT_ALL);
if (OBS_STATUS_OK != ret_status)
{
    printf("obs_initialize failed(%s).\n", obs_get_status_name(ret_status));
    return ret_status;
}
obs_deinitialize();
// Do not repeatedly call obs_initialize or obs_deinitialize, because these operations may lead to invalid memory access.
```

4.3 Configuring option

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

When the function of C SDK is called, **obs_options** must be input. You can use function **init_obs_options** to initialize configurations for **obs_options**, and configure AK, SK, endpoint, bucket, timeout period, and temporary authentication by using **obs_options**. **obs_options** consists of **obs_bucket_context** and **obs_http_request_option**. The following table describes the parameters that can be set.

Table 4-1 obs_options.obs_bucket_context parameters

Parameter	Description	Default Value	Recommended Value
host_name	The requested host name, which is the domain name (the endpoint) of the server where the requested resource is stored.	NULL	-
bucket_name	Name of the bucket where the operation is performed	NULL	-

Parameter	Description	Default Value	Recommended Value
protocol	The protocol used for sending requests. Possible values include HTTP and HTTPS. For security purposes, you are advised to use HTTPS.	HTTPS protocol: OBS_PROTOCOL_HTTPS	OBS_PROTOCOL_HTTPS
access_key	AK of OBS	NULL	-
secret_access_key	SK used for authentication. It can be used to sign a character string.	NULL	-
obs_storage_class	Set this parameter when the storage class needs to be configured in the PUT or POST request.	OBS Standard: OBS_STORAGE_CLASS_STANDARD	Default value
token	Security token of the temporary access key	NULL	-
bucket_type	Specifies whether a bucket is an object bucket or a parallel file system.	Bucket for object storage: OBS_BUCKET_OBJECT	-
bucket_list_type	Specifies whether to list all buckets, object buckets, or parallel file systems.	All buckets: OBS_BUCKET_LIST_ALL	-

Table 4-2 obs_options.obs_http_request_option parameters

Parameter	Description	Default Value	Recommended Value
connect_time	Timeout period for establishing an HTTP/HTTPS connection, in ms. The default value is **60,000**.	60000	10000 to 60000
max_connected_time	Timeout period (in seconds) of an HTTP/HTTPS request. The value 0 indicates that the link is never disconnected.	0	0
proxy_auth	Proxy authentication information, in the format of *Username*:*Password*	NULL	-
proxy_host	Proxy server	NULL	-

NOTE

If the network is unstable, you are advised to set larger values for **connect_time** and **max_connected_time**.

4.4 Configuring SDK Logging

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

The OBS C SDK log path is specified by the **LogPath** field in **OBS.ini**. By default, logs are stored in the **logs** directory at the same level as the **lib** directory of the C SDK dynamic library. **OBS.ini** must be in the same directory as **libeSDKLogAPI.so**.

The OBS C SDK allows you to use **set_obs_log_path** to specify a log path. This method has two parameters. The first parameter specifies a path and the second one determines how to set the path. If the second parameter is set to **True**, the SDK searches for **OBS.ini** in the path specified by the first parameter for log

configuration. If the second parameter is set to **False**, the SDK generates **OBS.ini** and log files in the path specified by the first parameter.

NOTE

- For details about SDK logging, see **Log Analysis**.

5 Bucket Management

5.1 Creating a Bucket

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can call **create_bucket** or **create_bucket_with_params** to create a bucket for object storage and use **create_pfs_bucket** to create a bucket used as a parallel file system.

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
canned_acl	obs_canned_acl. See **Managing Bucket ACLs**.	Mandatory	Access control policy
location_constraint	char *	Optional	The region where a bucket is to be created
handler	obs_response_handler *	Mandatory	Callback function

Field	Type	Mandatory or Optional	Description
callback_data	void *	Optional	Callback data
param	obs_create_bucket_params *	Mandatory	This field is used only in the **create_bucket_with_params** interface. The body contains: • **obs_canned_acl canned_acl**: permission control policy • **obs_az_redundancy az_redundancy**: multi-AZ or single-AZ – Single-AZ: **OBS_REDUNDANCY_1AZ** – 3-AZ: **OBS_REDUNDANCY_3AZ** • **const char *location_constraint**: region where the bucket resides. To create a 3-AZ bucket, you need to specify a region that supports 3-AZ storage.

Sample Code

The following code shows how to create a bucket for object storage:

```
static void test_create_bucket(obs_canned_acl canned_acl, char *bucket_name)
{
    // Create and initialize option.
    obs_options option;
    obs_status ret_status = OBS_STATUS_BUTT;
    init_obs_options(&option);

    option.bucket_options.host_name = "<your-endpoint;
    option.bucket_options.bucket_name = "<Your bucketname>";

// 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
```



```
    // Set the response callback function.
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // Create a bucket. For details about the predefined access policy, see section "Managing Bucket ACLs".
    create_bucket(&option, "<bucket ACL>", NULL, &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("create bucket successfully. \n");
    }
    else
    {
        printf("create bucket failed(%s).\n", obs_get_status_name(ret_status));
    }
}

The following code shows how to create a 3-AZ bucket:
static void test_create_3az_bucket(obs_canned_acl canned_acl, char *bucket_name)
{
    // Create and initialize option.
    obs_options option;
    obs_status ret_status = OBS_STATUS_BUTT;
    init_obs_options(&option);

    option.bucket_options.host_name = "<your-endpoint;
    option.bucket_options.bucket_name = "<Your bucketname>";
    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.     // Obtain an AK/SK pair on the management console.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

      // Set response callback function.
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // Create a bucket. For details about the predefined access policy, see section "Managing Bucket ACLs".
    obs_create_bucket_params create_param;
    create_param.canned_acl = canned_acl;
    create_param.location_constraint = "Region name that support 3AZ";
    create_param.az_redundancy = OBS_REDUNDANCY_3AZ;
    create_bucket_with_params(&option, &create_param, &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("create bucket with params successfully. \n");
    }
    else
    {
        printf("create bucket with params failed(%s).\n", obs_get_status_name(ret_status));
    }
}
The following code shows how to create a bucket used as a parallel file system:
static void test_create_pfs_bucket(obs_canned_acl canned_acl, char *bucket_name)
{
    // Create and initialize option.
    obs_options option;
    obs_status ret_status = OBS_STATUS_BUTT;
    init_obs_options(&option);

    option.bucket_options.host_name = "<your-endpoint;
    option.bucket_options.bucket_name = "<Your bucketname>";
    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.     // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html..
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
```

```
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // Set response callback function.
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // Create a bucket. For details about the predefined access policy, see section "Managing Bucket ACLs".
    create_pfs_bucket(&option, "<bucket ACL>", NULL, &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("create bucket successfully. \n");
    }
    else
    {
        printf("create bucket failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

NOTE

Bucket names are globally unique. Ensure that the bucket you create is named differently from any other bucket. A bucket name must comply with the following rules:

- Contains 3 to 63 characters, starts with a digit or letter, and supports only lowercase letters, digits, hyphens (-), and periods (.)
- Cannot be an IP address.
- Cannot start or end with a hyphen (-) or period (.).
- Cannot contain two consecutive periods (.), for example, **my..bucket**.
- Cannot contain periods (.) and hyphens (-) adjacent to each other, for example, **my-.bucket** or **my.-bucket**.
- If you create buckets of the same name in a region, no error will be reported and the bucket properties comply with those set in the first creation request.

The bucket created in the previous example is of the default ACL (**private**), in the OBS Standard storage class, and in the default location where the global domain resides.

NOTICE

- During bucket creation, if the endpoint you use corresponds to the default region CN North-Beijing1 (cn-north-1), specifying a region is not a must. If the endpoint you use corresponds to any other region, except the default one, you must set the region to the one that the used endpoint corresponds to. To view the valid regions, see **Regions and Endpoints**. For example, if the endpoint used for initialization is **obs.ap-southeast-1.myhuaweicloud.com**, you must set **Location** to **ap-southeast-1** when you create a bucket. Otherwise, status code 400 will be returned.

Creating a Bucket with Parameters Specified

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

```
obs_status ret_status = OBS_STATUS_BUTT;

// Create and initialize option.
obs_options option;
init_obs_options(&option);
option.bucket_options.host_name = "<your-endpoint>";
```



```
option.bucket_options.bucket_name = "<Your bucketname>";

// 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
// Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // Set response callback function.
obs_response_handler response_handler =
{
    0, &response_complete_callback
};
// Set the storage class for a bucket.
option.bucket_options.storage_class = "<Your bucket storage policy>";

// Create a bucket. The predefined access policy and location of the bucket can be set for the input parameter.
create_bucket(&option, "<bucket ACL>", "<Your bucket location>", &response_handler, &ret_status);
if (ret_status == OBS_STATUS_OK) {
    printf("create bucket successfully. \n");
}
else
{
    printf("create bucket failed(%s).\n", obs_get_status_name(ret_status));
}
```

5.2 Listing Buckets

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can call **list_bucket_obs** to list buckets.

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
handler	obs_list_service_obs_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample Code

```
static void test_list_bucket_obs()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
// 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    list_service_data data;
    memset_s(&data, sizeof(list_service_data), 0, sizeof(list_service_data));
    // Customize response callback function.
    obs_list_service_obs_handler listHandler =
    {
        {NULL,
        &list_bucket_complete_callback },
        &listServiceObsCallback
    };
    // List buckets.
    list_bucket_obs(&option,&listHandler,&data);
    if (data.ret_status == OBS_STATUS_OK)
    {
        printf("list bucket successfully. \n");
    }
    else
    {
        printf("list bucket failed(%s).\n", obs_get_status_name(data.ret_status));
    }
}
```

NOTE

Obtained bucket names are listed in the lexicographical order.

5.3 Deleting a Bucket

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can call **delete_bucket** to delete a bucket.

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample Code

```
static void test_delete_bucket(char *bucket_name)
{
    // Create and initialize option.
    obs_options option;
    obs_status  ret_status = OBS_STATUS_BUTT;
    init_obs_options(&option);

    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

// 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_response_handler response_handler =
    {
        NULL,
        &response_complete_callback
    };

    delete_bucket(&option, &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("delete bucket successfully. \n");
    }
    else
    {
        printf("delete bucket failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

NOTE

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

5.4 Checking Whether a Bucket Exists

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can call **obs_head_bucket** to identify whether a bucket exists.

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample Code

```
static void test_head_bucket(char *bucket_name)
{
    // Create and initialize option.
    obs_status ret_status = OBS_STATUS_BUTT;
    obs_options option;
    init_obs_options(&option);

    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

// 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // Check whether a bucket exists.
    obs_head_bucket(&option, &response_handler, &ret_status);
```



```
    if (ret_status == OBS_STATUS_OK)
    {
        printf("head bucket successfully. \n");
    }
    else
    {
        printf("head bucket failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

5.5 Managing Bucket ACLs

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

A bucket **ACL** can be configured in any of the following ways:

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

The following table lists the five permissions supported by OBS.

Permission	Description	Value in OBS C SDK
READ	A grantee with this permission for a bucket can obtain the list of objects in and metadata of the bucket. A grantee with this permission for an object can obtain the object content and metadata.	obs_permission.OBS_PERMISSION_READ
WRITE	A grantee with this permission for a bucket can upload, overwrite, and delete any object in the bucket. This permission is not applicable to objects.	obs_permission.OBS_PERMISSION_WRITE

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

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

Table 5-1 Pre-defined Access Control Policies

Permission	Description	Value in OBS C SDK
private	The owner of a bucket or object has the **FULL_CONTROL** permission for the bucket or object. Other users have no permission to access the bucket or object.	obs_canned_acl.OBS_CANNED_ACL_PRIVATE
public-read	If this permission is set for a bucket, everyone can obtain the list of objects, multipart uploads, and object versions in the bucket, as well as metadata of the bucket. If this permission is set for an object, everyone can obtain the content and metadata of the object.	obs_canned_acl.OBS_CANNED_ACL_PUBLIC_READ
public-read-write	If this permission is set for a bucket, everyone can obtain the object list in the bucket, multipart uploads in the bucket, metadata of the bucket; upload objects; delete objects; initialize multipart uploads; upload parts; combine parts; copy parts; and abort multipart uploads. If this permission is set for an object, everyone can obtain the content and metadata of the object.	obs_canned_acl.OBS_CANNED_ACL_PUBLIC_READ_WRITE

Permission	Description	Value in OBS C SDK
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.	obs_canned_acl.OBS_CANNED_ACL_PUBLIC_READ_DELIVERED
public-read-write-delivered	If this permission is set for a bucket, everyone can obtain the object list in the bucket, multipart tasks in the bucket, metadata of the bucket; upload objects; delete objects; initialize multipart uploads; upload parts; combine parts; copy parts; abort multipart uploads; and obtain content and metadata of objects in the bucket. This permission cannot be set for objects.	obs_canned_acl.OBS_CANNED_ACL_PUBLIC_READ_WRITE_DELIVERED

Specifying a Pre-defined Access Control Policy During Bucket Creation

Sample code:

```
static void test_create_bucket(obs_canned_acl canned_acl, char *bucket_name)
{
    // Create and initialize option.
    obs_options option;
    obs_status ret_status = OBS_STATUS_BUTT;
    init_obs_options(&option);

    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

// 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
```



```
    };
    // canned_acl specifies a pre-defined access control policy during bucket creation.
    create_bucket(&option, canned_acl, NULL, &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("create bucket successfully. \n");
    }
    else
    {
        printf("create bucket failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

Setting a Pre-defined Access Control Policy for a Bucket

The following code shows how to set a pre-defined access control policy for a bucket.

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
canned_acl	obs_canned_acl	Mandatory	For details about how to manage bucket access rights, see **Table 5-1** in **Managing Bucket ACLs**.
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample code:

```
void test_set_bucket_acl_byhead(char *bucket_name)
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);

    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

// 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
```



```
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
// Set a pre-defined access policy for a bucket.
    obs_canned_acl canned_acl = OBS_CANNED_ACL_PUBLIC_READ_WRITE;
    set_bucket_acl_by_head(&option, canned_acl, &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("set bucket acl by head successfully. \n");
    }
    else
    {
        printf("set bucket acl by head failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

Directly Setting the Bucket ACL

The following code shows how to directly set bucket access permission parameters:

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
aclinfo	manager_acl_info *	Mandatory	Structure for managing ACL permission information
aclinfo->object_info	obs_object_info *	Ignoring malicious programs	Object name and version number. For non-multi-version objects, set **version** to **0**.
aclinfo->owner_id	char *	Ignoring malicious programs	User account ID
aclinfo->acl_grant_count_return	int *	Mandatory	Pointer to the number of returned **aclinfo->acl_grants**
aclinfo->acl_grants	obs_acl_grant *	Mandatory	Pointer to the permission information structure. For details, see **Table 5-2**.

Field	Type	Mandatory or Optional	Description
aclinfo->object_delivered	obs_object_delivered	Optional	Indicates whether the object ACL inherits the ACL of the bucket. Valid values are **OBJECT_DELIVERED_TRUE** and **OBJECT_DELIVERED_FALSE**. The default value is **OBJECT_DELIVERED_TRUE**.
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Table 5-2 Description of the permission information structure **obs_acl_grant**

Field	Type	Description
grantee_type	obs_grantee_type	For details, see **Table 5-3**.
grantee.canonical_user.id	char	CanonicalUser ID.
permission	obs_permission	For details, see Managing Bucket ACLs.
bucket_delivered	obs_bucket_delivered	Indicates whether the bucket ACL is transferred to the object in the bucket. Valid values are **BUCKET_DELIVERED_TRUE** and **BUCKET_DELIVERED_FALSE**. The default value is **BUCKET_DELIVERED_FALSE**.

Table 5-3 Description of the authorized user type

Field	Description
obs_grantee_type.OBS_GRANTEE_TYPE_CANONICAL_USER	An OBS user. Bucket or object permissions can be granted to any user who has an OBS account. Authorized users can use the AK and SK to access OBS.

Field	Description
obs_grantee_type.OBS_GRANTEE_TYPE_ALL_USERS	All users can access buckets or objects, including anonymous users.

Sample code:

```
void test_set_bucket_acl(char *bucket_name)
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);

    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

// 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // Create and initialize object information.
    manager_acl_info aclinfo;
    init_acl_info(&aclinfo);
    //Set a bucket ACL.
    set_bucket_acl(&option, &aclinfo, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("set bucket acl successfully. \n");
    }
    else
    {
        printf("set bucket acl failed(%s).\n", obs_get_status_name(ret_status));
    }
    // Free memory.
    deinitialize_acl_info(&aclinfo);
}
```

NOTE

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.

Obtaining the Bucket ACL

You can call **get_bucket_acl** to obtain the bucket ACL. Sample code:

```
void test_get_bucket_acl(char *bucket_name)
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";
```



```
// 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_response_handler response_handler =
    {
        0,&response_complete_callback
    };
    // Apply for the ACL structure memory.
    manager_acl_info *aclinfo = malloc_acl_info();
    // Call the API for obtaining permissions.
    get_bucket_acl(&option, aclinfo, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {
        printf("get bucket acl: ----------");
        printf("%s\n", aclinfo->owner_id);
        if (aclinfo->acl_grant_count_return)
        {
            print_grant_info(*aclinfo->acl_grant_count_return, aclinfo->acl_grants);
        }
    }
    else
    {
        printf("get bucket acl failed(%s).\n", obs_get_status_name(ret_status));
    }
    // Free memory.
    free_acl_info(&aclinfo);
}
```

5.6 Obtaining Bucket Storage Information

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

The storage information about a bucket includes the number of objects in and the used capacity of the bucket. You can call **get_bucket_storage_info** to obtain the bucket storage information.

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter

Field	Type	Mandatory or Optional	Description
capacity_length	int	Mandatory	Cache size of the used capacity
capacity	char *	Mandatory	Used capacity
object_number_length	int	Mandatory	Cache size of the object number
object_number	char *	Mandatory	Cache of the object number
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample Code

```
static void test_get_bucket_storage_info(char *bucket_name)
{
    // Create and initialize option.
    obs_options option;
    obs_status ret_status = OBS_STATUS_BUTT;
    init_obs_options(&option);

    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";
// 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Define cache of the bucket capacity and cache of the object number.
    char capacity[OBS_COMMON_LEN_256] = {0};
    char obj_num[OBS_COMMON_LEN_256] = {0};

    // Set response callback function.
    obs_response_handler response_handler =
    {
        NULL,
        &response_complete_callback
    };

    // Obtain bucket storage information.
    get_bucket_storage_info(&option, OBS_COMMON_LEN_256, capacity, OBS_COMMON_LEN_256, obj_num,
        &response_handler, &ret_status);

    if (ret_status == OBS_STATUS_OK) {
        printf("get_bucket_storage_info success,bucket=%s objNum=%s capacity=%s\n",
            bucket_name, obj_num, capacity);
```



```
    }
    else
    {
        printf("head bucket failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

5.7 Bucket Quota

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

Setting a Bucket Quota

You can use the **set_bucket_quota** function to set the bucket quota. The following table describes the parameters.

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
storage_quota	uint64_t	Mandatory	Quota size, in bytes.
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample code:

```
static void test_set_bucket_quota( char *bucket_name)
{
    obs_status ret_status = OBS_STATUS_BUTT;
    uint64_t bucketquota = 104857600;

    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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
```



```
ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // Set bucket quota.
    set_bucket_quota(&option, bucketquota, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {
        printf("set bucket quota successfully. \n");
    }
    else
    {
        printf("set bucket quota failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

NOTE

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

Obtaining a Bucket Quota

You can use the **get_bucket_quot** function to obtain bucket quotas. The following table describes the parameters.

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
storagequota_return	uint64_t *	Mandatory	The obtained quota size, in bytes.
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample code:

```
static void test_get_bucket_quota( char *bucket_name)
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
```

```
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // Obtain the bucket quota.
    uint64_t bucketquota = 0;
    get_bucket_quota(&option, &bucketquota, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("Bucket=%s  Quota=%lu \n get bucket quota successfully. \n ",
            bucket_name, bucketquota);
    }
    else
    {
        printf("get bucket quota failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

5.8 Storage Class

NOTICE

If you have any questions during the development, post them on the **Issues** page of GitHub.

OBS allows you to set storage classes for buckets. The storage class of an object defaults to be that of its residing bucket. There are three types of storage class for buckets, as described in the following table, catering to various storage performance and cost requirements.

Type	Description	Value in OBS C 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.	OBS_STORAGE_CLASS_STANDARD
OBS Infrequent Access	Is applicable to storing semi-frequently accessed (less than 12 times a year) data requiring quick response.	OBS_STORAGE_CLASS_STANDARD_IA

Type	Description	Value in OBS C SDK
OBS Archive	Is applicable to archiving rarely-accessed (once a year) data.	OBS_STORAGE_CLASS_GLACIER

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

Setting the Storage Class for a Bucket

You can use **set_bucket_storage_class_policy** to set the bucket storage class. The following table describes the parameters.

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
storage_class_policy	obs_storage_class	Mandatory	Storage class
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample code:

```
static void test_set_bucket_storage_class(char *bucket_name,
                obs_storage_class storage_class_policy)
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_response_handler response_handler =
```



```
    {
        0, &response_complete_callback
    };

    set_bucket_storage_class_policy(&option, storage_class_policy,
                &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("set bucket storage class successfully. \n");
    }
    else
    {
        printf("set bucket storage class failed(%s).\n",
                            obs_get_status_name(ret_status));
    }
}
```

Obtaining the Storage Class of a Bucket

You can use **get_bucket_storage_class_policy** to obtain the bucket storage class. The following table describes the parameters.

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
handler	obs_get_bucket_storage_class_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample code:

```
static void test_get_bucket_storage_class(char *bucket_name)
{
    // Create and initialize option.
    obs_status ret_status = OBS_STATUS_BUTT;
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_get_bucket_storage_class_handler getBucketStorageResponse =
    {
```



```
        {0, &response_complete_callback},
        &get_bucket_storageclass_handler
    };
    //Obtain the bucket storage class.
    get_bucket_storage_class_policy(&option, &getBucketStorageResponse, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {
        printf("get bucket storage class successfully.\n");
    }
    else
    {
        printf("get bucket storage class failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

6 Uploading an Object

6.1 Performing a Streaming Upload

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

In a streaming upload, your data is transferred to **callback_data**, and the callback function **put_object_data_callback** defined in the **obs_put_object_handler** structure is used to process the transferred callback data. Call **put_object** to implement streaming upload.

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
key	char *	Mandatory	Object name
content_length	uint64_t	Mandatory	Length of the object content
put_properties	obs_put_properties*	Mandatory	Properties of the uploaded object

Field	Type	Mandatory or Optional	Description
encryption_params	server_side_encryption_params *	Optional	Encryption setting of the uploaded object
handler	obs_put_object_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample Code

```
static void test_put_object_from_buffer()
{
    // Buffer to be uploaded
   char *buffer = "abcdefg";
    // Length of the buffer to be uploaded
   int buffer_size = strlen(buffer);
    // Name of an object to be uploaded
   char *key = "put_buffer_test";

   // Initialize option.
   obs_options option;
   init_obs_options(&option);
   option.bucket_options.host_name = "<your-endpoint>";
   option.bucket_options.bucket_name = "<Your bucketname>";

   // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
   // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
   option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
   option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
   option.bucket_options.certificate_info = "<Your certificate>";
   // Initialize the properties of an object to be uploaded.
   obs_put_properties put_properties;
   init_put_properties(&put_properties);

   // Initialize the structure for storing uploaded data.
   put_buffer_object_callback_data data;
   memset(&data, 0, sizeof(put_buffer_object_callback_data));
   // Assign the buffer value to the structure.
   data.put_buffer = buffer;
   // Set buffersize.
   data.buffer_size = buffer_size;

   // Set callback function.
   obs_put_object_handler putobjectHandler =
   {
      { &response_properties_callback, &put_buffer_complete_callback },
        &put_buffer_data_callback
   };

   put_object(&option, key, buffer_size, &put_properties,0,&putobjectHandler,&data);
   if (OBS_STATUS_OK == data.ret_status) {
      printf("put object from buffer successfully. \n");
```



```
    }
    else
    {
        printf("put object from buffer failed(%s).\n",
                        obs_get_status_name(data.ret_status));
    }
}
```

NOTE

- The content to be uploaded cannot exceed 5 GB.
- The fifth parameter in **put_object** is used to provide the server-side encryption. For details, see section Server-Side Encryption.

6.2 Performing a File-Based Upload

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

File-based upload uses local files as the data source of objects. For parameters, see **Performing a Streaming Upload**. Sample code:

```
static void test_put_object_from_file()
{
    // Name of an object to be uploaded
    char *key = "put_file_test";
    // File to be uploaded
    char file_name[256] = "./put_file_test.txt";
    uint64_t content_length = 0;
    // Initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    option.bucket_options.certificate_info = "<Your certificate>";
    // Initialize the properties of an object to be uploaded.
    obs_put_properties put_properties;
    init_put_properties(&put_properties);

    // Initialize the structure for storing uploaded data.
    put_file_object_callback_data data;
    memset(&data, 0, sizeof(put_file_object_callback_data));
    // Open the file and obtain the file length.
    content_length = open_file_and_get_length(file_name, &data);
    // Set callback function.
    obs_put_object_handler putobjectHandler =
    {
        { &response_properties_callback, &put_file_complete_callback },
        &put_file_data_callback
    };

    put_object(&option, key, content_length, &put_properties, 0, &putobjectHandler, &data);
```



```
    if (OBS_STATUS_OK == data.ret_status) {
        printf("put object from file successfully. \n");
    }
    else
    {
        printf("put object failed(%s).\n",
               obs_get_status_name(data.ret_status));
    }
}
```

NOTE

- The content to be uploaded cannot exceed 5 GB.
- When uploading file streams, you must open files in **rb** or **rb+** mode.

6.3 Creating a Folder

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

OBS does not involve folders like in a file system. 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.

```
static void test_creat_folder()
{
    char *key = "YourFolder/";
    // Initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    option.bucket_options.uri_style = OBS_URI_STYLE_PATH;
    temp_auth_configure tempauth;
    tempAuthResult  ptrResult;
    memset(&ptrResult,0,sizeof(tempAuthResult));
    // Initialize the properties of an object to be uploaded.
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    // Initialize the structure for storing uploaded data.
    put_file_object_callback_data data;
    memset(&data, 0, sizeof(put_file_object_callback_data));
    // Set callback function.
    obs_put_object_handler putobjectHandler =
    {
        { &response_properties_callback, &put_file_complete_callback },
        &put_file_data_callback
    };
```



```
    put_object(&option, key, 0, &put_properties, 0, &putobjectHandler, &data);
    if (OBS_STATUS_OK == data.ret_status)
    {
        printf("put object from file successfully. \n");
    }
    else
    {
        printf("put object failed(%s).\n",
            obs_get_status_name(data.ret_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.

6.4 Setting Object Properties

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

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. Object attributes can be uploaded in several modes (streaming upload, file-based upload, multipart upload, browser-based upload) or be configured when **Copying an Object**. Parameters to set object properties are in the **obs_put_properties** structure.

The following table describes object properties.

Property Name	Description	Default Value
Object length (content_length)	Indicates the object length. If the object length exceeds the stream or file length, the object will be truncated.	Actual length of the stream or file
Object MIME type (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.	N/A

Property Name	Description	Default Value
MD5 value of the object (md5)	Indicates the base64-encoded digest of the object data. It is provided for the OBS server to verify data integrity.	N/A
Storage class	Indicates the storage class of the object. Different storage classes meet different needs for storage performance and costs. The value defaults to be the same as the object's residing bucket and can be changed.	N/A
Customized metadata	Indicates the user-defined description of properties of the object uploaded to the bucket. It is used to facilitate the customized management on the object.	N/A

Setting the MIME Type for an Object

You can set the object MIME type by assigning a value to **content_type** in the **obs_put_properties** structure. The following uses file upload as an example to describe how to set the MIME type of an object:

```
static void test_put_object_from_file2()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Initialize put_properties which can be used to set object properties.
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    // Set the MIME type.
    put_properties.content_type = "text/html";
    // Callback data
    put_file_object_callback_data data;
    memset(&data, 0, sizeof(put_file_object_callback_data));
    // Read the file to be uploaded to the callback data.
    data.infile = 0;
    data.noStatus = 1;
    content_length = read_bytes_from_file("<Uploaded filename>", &data);
```



```
    // Callback function
    obs_put_object_handler putobjectHandler =
    {
        { &response_properties_callback, &response_complete_callback },
        &put_object_data_callback
     };
    // Upload data streams.
    put_object(&option,"<object key>", content_length, &put_properties, 0, &putobjectHandler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("put object from file successfully. \n");
    }
    else
    {
        printf("put object failed(%s).\n",
               obs_get_status_name(data.ret_status));
    }
}
```

NOTE

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

Setting the Storage Class for an Object

You can set the object storage class by assigning a value to **storage_class** in **obs_bucket_context**. Sample code:

```
static void test_put_object_from_file3()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set the storage class to Archive.
    option.bucket_options.storage_class = OBS_STORAGE_CLASS_GLACIER;
    // Initialize put_properties which can be used to set object properties.
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    // Callback data
    put_file_object_callback_data data;
    memset(&data, 0, sizeof(put_file_object_callback_data));
    // Read the file to be uploaded to the callback data.
    data.infile = 0;
    data.noStatus = 1;
    content_length = read_bytes_from_file("<Uploaded filename>", &data);
    // Callback function
    obs_put_object_handler putobjectHandler =
    {
        { &response_properties_callback, &response_complete_callback },
        &put_object_data_callback
    };
     // Upload data streams.
    put_object(&option,"<object key>", content_length, &put_properties, 0, &putobjectHandler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("put object from file successfully. \n");
    }
```

```
    else
    {
        printf("put object failed(%s).\n",
            obs_get_status_name(data.ret_status));
    }

}
```

NOTE

- The storage class of the objects in a bucket is the same as that of the bucket.
- There are three object storage classes. Their meanings are the same as those described in **Storage Class**.
- Before downloading an Archive object, you must restore it.

Customizing Metadata for an Object

You can set the object customized metadata by assigning a value to **meta_data** in **obs_put_properties**. Sample code:

```
static void test_put_object_from_file4()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Initialize put_properties which can be used to set object properties.
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    // Customize metadata.
    obs_name_value matadata;
    matadata.name ="property1";
    matadata.value ="property-value1";
    put_properties.meta_data = matadata;
    // Callback data
    put_file_object_callback_data data;
    memset(&data, 0, sizeof(put_file_object_callback_data));
    // Read the file to be uploaded to the callback data.
    data.infile = 0;
    data.noStatus = 1;
    content_length = read_bytes_from_file("<Uploaded filename>", &data);
    // Callback function
    obs_put_object_handler putobjectHandler =
    {
        { &response_properties_callback, &response_complete_callback },
        &put_object_data_callback
     };
     // Upload data streams.
    put_object(&option,"<object key>", content_length, &put_properties, 0, &putobjectHandler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("put object from file successfully. \n");
    }
    else
    {
        printf("put object failed(%s).\n",
              obs_get_status_name(data.ret_status));
    }
}
```


NOTE

- In the preceding sample code for customizing metadata for an object, a user has defined a metadata named property1 and whose value is property-value1.
- An object can have multiple pieces of metadata. The total metadata size cannot exceed 8 KB.
- You can obtain the customized metadata of an object by using **get_object_metadata**. For details, see **Obtaining Object Properties**.
- When you use **get_object** to download an object, its customized metadata will also be downloaded.

6.5 Performing a Multipart Upload

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

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

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

Multipart upload consists of three phases:

Step 1 Initiate a multipart upload (**initiate_multi_part_upload**).

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

Step 3 Combine parts (**complete_multi_part_upload**) or abort the multipart upload (**abort_multi_part_upload**).

----End

Initiating a Multipart Upload

Before using a multipart upload, you need to first let OBS initiate it. This operation will return a globally unique identifier (**upload_id**) 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.

Note that in a multipart upload, the object properties (such as the ACL and expiration time) can only be configured in the initiation phase.

You can use **initiate_multi_part_upload** to initiate a multipart upload. The following table describes related parameters.

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
key	char *	Mandatory	Object name
upload_id_return_size	int	Mandatory	Size of the cache for storing multipart upload IDs
upload_id_return	char *	Mandatory	Cache of the multipart upload ID
put_properties	obs_put_properties*	Optional	Properties of the uploaded object
encryption_params	server_side_encryption_params *	Optional	Configuring server-side encryption
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

```
static void test_initiate_multi_part_upload()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Define the cache and size of multipart upload IDs.
    char upload_id[OBS_COMMON_LEN_256] = {0};
    int upload_id_size = OBS_COMMON_LEN_256;
    // Set response callback function.
    obs_response_handler handler =
    {
        &response_properties_callback,
        &response_complete_callback
    };
    // Initiate a multipart upload. Set upload_id below to the value of upload_id_return listed in the parameter table above.
```



```
    initiate_multi_part_upload(&option, "<object key>", upload_id_size, upload_id, NULL, NULL, &handler,
&ret_status);
    if (OBS_STATUS_OK == ret_status)
    {
        printf("test init upload part successfully. uploadId= %s\n", upload_id);
    }
    else
    {
        printf("test init upload part faied(%s).\n", obs_get_status_name(ret_status));
    }
}
```

NOTE

- In **obs_put_properties** structure, you can specify the MIME type and customized metadata for the object.
- **upload_id** of the multipart upload returned by **initiate_multi_part_upload** will be used in follow-up operations.

Uploading Parts

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

You can upload a part by using **upload_part**. The following table describes the parameters.

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
key	char *	Mandatory	Object name
upload_part_info	obs_upload_part_info *	Mandatory	Information about the uploading part
upload_part_info->part_number	unsigned int	Mandatory	ID of the uploading part The value is an integer from 1 to 10000.
upload_part_info->upload_id	char *	Mandatory	Task ID of a multipart upload

Field	Type	Mandatory or Optional	Description
content_length	uint64_t	Mandatory	Length of the uploaded content
put_properties	obs_put_properties*	Optional	Properties of the uploaded object
encryption_params	server_side_encryption_params *	Optional	Configuring server-side encryption
handler	obs_upload_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample code:

```
static void test_upload_part()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

// 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Define the slice size by 5 MB.
    uint64_t uploadSliceSize =5L * 1024 * 1024;
    // Define and initialize the size of the uploading part.
    uint64_t uploadSize = uploadSliceSize;
    // Define and Initialize the length variable of the uploading part.
    uint64_t filesize = 0;
    // Initialize put_properties.
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    // Callback function
    obs_upload_handler Handler =
    {
        {&response_properties_callback, &response_complete_callback},
        &test_upload_file_data_callback
    };
    // Initialize callback data.
    test_upload_file_callback_data data;
    memset(&data, 0, sizeof(test_upload_file_callback_data));
    filesize = get_file_info(filename,&data);
    data.noStatus = 1;
    data.part_size = uploadSize;
    data.part_num = (filesize % uploadSize == 0) ? (filesize / uploadSize) : (filesize / uploadSize +1);
```



```
    // Upload the first part.
    uploadPartInfo.part_number=1;
    uploadPartInfo.upload_id = "<upload id>";
    data.start_byte  = 0;
    upload_part(&option,key,&uploadPartInfo,uploadSize,
                                &put_properties,0,&Handler,&data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("test upload part 1 successfully. \n");
    }
    else
    {
        printf("test upload part 1 faied(%s).\n", obs_get_status_name(data.ret_status));
    }
    // Upload the second part.
    uploadPartInfo.part_number=2;
    uploadPartInfo.upload_id = "<upload id>";
    filesize = get_file_info(filename,&data);
    uploadSize =filesize - uploadSize;
    data.part_size = uploadSize;
    data.start_byte = uploadSliceSize;
    fseek(data.infile, data.start_byte, SEEK_SET);
    upload_part(&option,key,&uploadPartInfo,uploadSize, &put_properties,0,&Handler,&data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("test upload part 2 successfully. \n");
    }
    else
    {
        printf("test upload part 2 faied(%s).\n", obs_get_status_name(data.ret_status));
    }

}
```

NOTE

- Except the part last uploaded, other parts must be larger than 100 KB. Part sizes will not be verified during upload because which one is last uploaded is not identified until parts are combined.
- OBS will return ETags (MD5 values) of the received parts to users.
- To ensure data integrity, set the value of MD5 and add the MD5 value to the **Content-MD5** request header. The OBS server will compare the MD5 value contained by each part and that calculated by SDK to verify the data integrity.
- You can call **put_properties.md5** to set the MD5 value of the uploaded data directly.
- Part numbers range from 1 to 10000. If a part number exceeds 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.

Complete Multipart Upload

After all parts are uploaded, call the API for combining parts to form 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 combine parts by using **complete_multi_part_upload**. The following table describes the parameters.

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
key	char *	Mandatory	Object name
upload_id	char *	Mandatory	Indicates a multipart upload.
part_number	unsigned int	Mandatory	Number of segments, **complete_upload_Info** array length
complete_upload_Info	obs_complete_upload_Info *	Mandatory	Parts information array
complete_upload_Info->part_number	unsigned int	Mandatory	Part number
complete_upload_Info->etag	char *	Mandatory	ETag value of a part
put_properties	obs_put_properties*	Optional	Properties of the uploaded object
handler	obs_complete_multi_part_upload_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample code:

```
static void test_complete_upload(char *filename, char *key)
{
    obs_status ret_status = OBS_STATUS_BUTT;

    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    //Read the AK/SK from environment variables.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // Initialize the put_properties structure.
```



```
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    // Set part information.
    char *uploadId = "<upload id>";
    obs_complete_upload_Info info[2];
    info[0].part_number="1";
    info[0].etag="65fe0e161b35c8deead213871033f7fa";
    info[1].part_number="2";
    info[1].etag="0433d5ffc28450be3b6cf25ab8955267";
     // Set response callback function.
    obs_complete_multi_part_upload_handler Handler =
    {
       {&response_properties_callback,
        &response_complete_callback},
       &CompleteMultipartUploadCallback
    };
    // Combine parts.
    complete_multi_part_upload(&option,key,uploadId,number,info,&putProperties,
              &Handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
       printf("test complete upload successfully. \n");
    }
    else
    {
       printf("test complete upload faied(%s).\n", obs_get_status_name(ret_status));
    }
}
```

NOTE

- In the preceding code, the **info** structure array indicates the list of part numbers and ETags of uploaded parts.
- Part numbers can be inconsecutive.

Concurrently Uploading Parts

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

```
static void test_concurrent_upload_part(char *filename, char *key, uint64_t uploadSliceSize)
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    //Read the AK/SK from environment variables.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    char *concurrent_upload_id;
    uint64_t uploadSize = uploadSliceSize;
    uint64_t filesize =0;
    // Initialize the put_properties structure.
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    // Large file information: file pointer, file size, number of parts determined by part size
    test_upload_file_callback_data data;
    memset(&data, 0, sizeof(test_upload_file_callback_data));
    filesize = get_file_info(filename,&data);
    data.noStatus = 1;
    data.part_size = uploadSize;
    data.part_num = (filesize % uploadSize == 0) ? (filesize / uploadSize) : (filesize / uploadSize +1);
    // Initialize the callback function of the uploading part.
    obs_response_handler Handler =
```

```
    {
        &response_properties_callback, &response_complete_callback
    };
    // Callback function of the combined parts.
    obs_complete_multi_part_upload_handler complete_multi_handler =
    {
        {&response_properties_callback,
         &response_complete_callback},
        &CompleteMultipartUploadCallback
    };
    // Initialization of the uploading part returns uploadId: uploadIdReturn.
    char uploadIdReturn[256] = {0};
    int upload_id_return_size = 255;
    initiate_multi_part_upload(&option,key,upload_id_return_size,uploadIdReturn, &putProperties,
            0,&Handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("test init upload part return uploadIdReturn(%s). \n", uploadIdReturn);
        strcpy(concurrent_upload_id,uploadIdReturn);
    }
    else
    {
        printf("test init upload part faied(%s).\n", obs_get_status_name(ret_status));
    }
    // Concurrent upload
    test_concurrent_upload_file_callback_data *concurrent_upload_file;
    concurrent_upload_file =(test_concurrent_upload_file_callback_data *)malloc(
                    sizeof(test_concurrent_upload_file_callback_data)*(data.part_num+1));
    if(concurrent_upload_file == NULL)
    {
         printf("malloc test_concurrent_upload_file_callback_data failed!!!");
         return ;
    }
    test_concurrent_upload_file_callback_data *concurrent_upload_file_complete =
                              concurrent_upload_file;
    start_upload_threads(data, concurrent_upload_id,filesize, key, option, concurrent_upload_file_complete);
     // Combine parts.
    obs_complete_upload_Info *upload_Info = (obs_complete_upload_Info *)malloc(
                sizeof(obs_complete_upload_Info)*data.part_num);
    for(i=0; i<data.part_num; i++)
    {
        upload_Info[i].part_number = concurrent_upload_file_complete[i].part_num;
        upload_Info[i].etag = concurrent_upload_file_complete[i].etag;
    }
    complete_multi_part_upload(&option, key, uploadIdReturn,
data.part_num,upload_Info,&putProperties,&complete_multi_handler,&ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("test complete upload successfully. \n");
    }
    else
    {
        printf("test complete upload faied(%s).\n", obs_get_status_name(ret_status));
    }
    if(concurrent_upload_file)
    {
        free(concurrent_upload_file);
        concurrent_upload_file = NULL;
    }
    if(upload_Info)
    {
        free(upload_Info);
        upload_Info = NULL;
    }
}
```


6.6 Performing a Multipart Copy

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

As a special case of multipart upload, multipart copy implements multipart upload by copying the whole or part of an object in a bucket. You can copy parts by using **copy_part**. The following table describes the parameters.

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
key	char *	Mandatory	Object name
object_info	obs_copy_destination_object_info *	Mandatory	Indicates a multipart upload.
object_info->destination_bucket	char *	Mandatory	Bucket where the target object is located
object_info->destination_key	char *	Mandatory	Target object name
object_info->last_modified_return	int64_t *	Mandatory	Latest time when the object was modified
object_info->etag_return_size	int	Mandatory	Cache size of ETag
object_info->etag_return	char *	Mandatory	Cache of ETag
copypart	obs_upload_part_info *	Mandatory	Information about the part to be uploaded
copypart->part_number	unsigned int	Mandatory	ID of the part to be uploaded

Field	Type	Mandatory or Optional	Description
copypart->upload_id	char *	Mandatory	Task ID of a multipart upload
put_properties	obs_put_properties*	Optional	Properties of the uploaded object
encryption_params	server_side_encryption_params *	Optional	Configuring server-side encryption
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample code:

```
static void test_copy_part()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // SSE-KMS encryption
    server_side_encryption_params encryption_params;
    memset(&encryption_params, 0, sizeof(server_side_encryption_params));
    // ETag value returned after the part is copied
    char etagreturn[256] ={0};

    char *key= "<Source object key>"
    // Define the information of copied parts.
    obs_copy_destination_object_info object_info;
    memset(&object_info, 0, sizeof(obs_copy_destination_object_info));
    object_info.destination_bucket = "<Your destination bucketname>";
    object_info.destination_key = "<Your destination object key>";
    object_info.etag_return = etagreturn;
    object_info.etag_return_size = 256;
    obs_upload_part_info copypart;
    memset(&copypart, 0, sizeof(obs_upload_part_info));
     // Set response callback function.
    obs_response_handler responseHandler =
    {
        &response_properties_callback,
        &response_complete_callback
```



```
    };
    // Copy the first part.
    copypart.part_number = "1";
    copypart.upload_id = "<upload id>";
    copy_part(&option, key, &object_info, &copypart,
              &putProperties,&encryption_params,&responseHandler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf(" copy part 1 successfully. \n");
    }
    else
    {
        printf("copy part 1 failed(%s).\n", obs_get_status_name(ret_status));
    }
    // Copy the second part.
    copypart.part_number = "2";
    copypart.upload_id = "<upload id>";
    copy_part(&option, key, &object_info, &copypart,
              &putProperties,&encryption_params,&responseHandler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf(" copy part 2 successfully. \n");
    }
    else
    {
        printf("copy part 2 failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

6.7 Performing a Resumable Upload

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

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

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

Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
key	char *	Mandatory	Object name
upload_file_config	obs_upload_file_configuration *	Mandatory	For details about the configuration of the file to be uploaded, see the following table.
encryption_params	server_side_encryption_params *	Optional	Encryption setting of the uploaded object
handler	obs_upload_file_response_handler *	Mandatory	Callback body. All member variables are pointers to the callback function.
callback_data	void *	Optional	Callback data

The following table describes the structure of **obs_upload_file_configuration**.

Member Name	Type	Mandatory or Optional	Description
upload_file	char *	Mandatory	Local file to be uploaded
part_size	uint64_t	Mandatory	Part size, in bytes. The value ranges from 100 KB to 5 GB. The default value is 5 MB.
check_point_file	char *	Mandatory	File used to record the upload progress. This parameter is effective only in the resumable upload mode. If the value is null, the file is in the same directory as the local file to be uploaded.

Member Name	Type	Mandatory or Optional	Description
enable_check_point	int	Mandatory	Whether to enable the resumable upload mode. The default value is **0**, which indicates that this mode is disabled.
task_num	int	Mandatory	Maximum number of parts that can be concurrently uploaded. The default value is **1**.

Sample Code

Sample code:

```
void uploadFileResultCallback(obs_status status,
                              char *resultMsg,
                              int partCountReturn,
                              obs_upload_file_part_info * uploadInfoList,
                              void *callbackData);
//Declare the callback function.
static void test_upload_file()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Initialize the put_properties structure.
    obs_put_properties put_properties;
    init_put_properties(&put_properties);

    obs_upload_file_configuration uploadFileInfo;

memset_s(&uploadFileInfo,sizeof(obs_upload_file_configuration),0,sizeof(obs_upload_file_configuration));
    uploadFileInfo.check_point_file = 0;
    uploadFileInfo.enable_check_point = 1;
    uploadFileInfo.part_size = "<part size>";
    uploadFileInfo.task_num = "<task num>";
    uploadFileInfo.upload_file = "<upload filename>";
    uploadFileInfo.put_properties = &put_properties;

    // Callback function
    obs_upload_file_response_handler Handler =
    {
        {&response_properties_callback, &response_complete_callback_for_multi_task},
```



```
    &uploadFileResultCallback
  };
  initialize_break_point_lock();
  upload_file(&option, "<Your Key>", 0, &uploadFileInfo, Null, &Handler, &ret_status);
  deinitialize_break_point_lock();
  if (OBS_STATUS_OK == ret_status) {
    printf("test upload file successfully. \n");
  }
  else
  {
    printf("test upload file faied(%s).\n", obs_get_status_name(ret_status));
  }
}
//uploadFileResultCallback is used as an example here. The printf statements can be replaced with custom logging print statements.
void uploadFileResultCallback(obs_status status,
                              char *resultMsg,
                              int partCountReturn,
                              obs_upload_file_part_info * uploadInfoList,
                              void *callbackData)
{
  int i=0;
  obs_upload_file_part_info * pstUploadInfoList = uploadInfoList;
  printf("status return is %d(%s)\n",status,obs_get_status_name(status));
  printf("%s",resultMsg);
  printf("partCount[%d]\n",partCountReturn);
  for(i=0;i<partCountReturn;i++)
  {
    printf("partNum[%d],startByte[%llu],partSize[%llu],status[%d]\n",
    pstUploadInfoList->part_num,
    pstUploadInfoList->start_byte,
    pstUploadInfoList->part_size,
    pstUploadInfoList->status_return);
    pstUploadInfoList++;
  }
  if (callbackData) {
    obs_status* retStatus = (obs_status*)callbackData;
    (*retStatus) = status;
  }
}
```

NOTE

- The API for resumable upload, which is implemented based on **Performing a Multipart Upload**, is an encapsulated and enhanced version of multipart upload.
- This API saves resources and improves efficiency upon the re-upload, and speeds up the upload process by concurrently uploading parts. Because this API is transparent to users, users are free from concerns about internal service details, such as the creation and deletion of checkpoint files, division of objects, and concurrent upload of parts.
- **enable_check_point**: The value 0 indicates that the resumable upload mode is disabled. In this case, the resumable upload API functions as an encapsulated version of multipart upload and does not generate checkpoint files. The value 1 indicates that the resumable upload mode is enabled.

6.8 Performing an Appendable Upload

NOTICE

If you have any questions during the development, post them on the **Issues** page of GitHub.

The API **append_object** shares the same parameters and usage as that of **put_object**. The only difference is that the start position of the write object is added.

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
key	char *	Mandatory	Object name
content_length	uint64_t	Mandatory	Length of the object content
position	char *	Mandatory	Start position of the appended write object
put_properties	obs_put_properties*	Mandatory	Properties of the uploaded object
encryption_params	server_side_encryption_params *	Optional	Encryption setting of the uploaded object
handler	obs_append_object_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample Code

```
static void test_append_object_from_buffer()
{
    // Buffer to be uploaded
    char *buffer = "abcdefg";
    // Length of the buffer to be uploaded
    int buffer_size = strlen(buffer);
    // Name of an object to be uploaded
    char *key = "put_buffer_test";
    char * position = "0";
    // Initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
```



```
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    option.bucket_options.certificate_info = "<Your certificate>";
    // Initialize the properties of an object to be uploaded.
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    // Initialize the structure for storing uploaded data.
    put_buffer_object_callback_data data;
    memset(&data, 0, sizeof(put_buffer_object_callback_data));
    // Assign the buffer value to the structure.
    data.put_buffer = buffer;
    // Set buffersize.
    data.buffer_size = buffer_size;
    // Set callback function.
    obs_append_object_handler putobjectHandler =
    {
        { &response_properties_callback, &response_complete_callback},
          &put_buffer_data_callback
    };
    append_object(&option, key, buffer_size, position, &put_properties,
    0,&putobjectHandler,&data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("put object from buffer successfully. \n");
    }
    else
    {
        printf("put object from buffer failed(%s).\n", obs_get_status_name(data.ret_status));
    }
}
```

NOTE

- Objects uploaded using **put_object**, referred to as normal objects, can overwrite objects uploaded using **append_object**, referred to as appendable objects. Data cannot be appended to an appendable object anymore once the object has been overwritten by a normal object.
- When you upload an object for the first time in appendable mode, an error will be reported (HTTP status code **409**) if a common object with the same name is already present.
- The ETag returned for each append 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 10,000 times of appendable uploads can be performed on a single object.
- The sixth parameter in **append_object** is used to provide the server-side encryption. For details, see section Server-Side Encryption.

6.9 Performing a Modification

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

The modification API **modify_object** applies only to buckets used as parallel file systems. This API shares the same parameters and usage as that of **put_object**. The only difference is that the start position of the write object is added.

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
key	char *	Mandatory	Object name
content_length	uint64_t	Mandatory	Length of the object content
position	char *	Mandatory	Start position of the object to be modified
put_properties	obs_put_properties*	Mandatory	Properties of the uploaded object
encryption_params	server_side_encryption_params *	Optional	Encryption setting of the uploaded object
handler	obs_modify_object_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample Code

```
static void test_modify_object_from_buffer()
{
    // Buffer to be uploaded
    char *buffer = "abcdefg";
    // Length of the buffer to be uploaded
    int buffer_size = strlen(buffer);
    // Name of an object to be uploaded
    char *key = "put_buffer_test";
    char * position = "0";
    // Initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    option.bucket_options.certificate_info = "<Your certificate>";
```



```
    // Initialize the properties of an object to be uploaded.
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    // Initialize the structure for storing uploaded data.
    put_buffer_object_callback_data data;
    memset(&data, 0, sizeof(put_buffer_object_callback_data));
    // Assign the buffer value to the structure.
    data.put_buffer = buffer;
    // Set buffersize.
    data.buffer_size = buffer_size;
    // Set callback function.
    obs_modify_object_handler modify0bjectHandler =
    {
        { &response_properties_callback, &put_buffer_complete_callback },
          &put_buffer_data_callback
    };
    modify_object(&option, key, buffer_size, position, &put_properties,
    0, &modify0bjectHandler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("modify object from buffer successfully. \n");
    }
    else
    {
        printf("modify object from buffer failed(%s).\n", obs_get_status_name(data.ret_status));
    }
}
```

NOTE

- **modify_object** applies only to buckets used as parallel file systems.
- When you call **modify_object** for the first time, if the object to be modified does not exist, an error (HTTP status code: 404) will be reported.
- The sixth parameter in **modify_object** is used to provide the server-side encryption. For details, see section Server-Side Encryption.

7 Downloading an Object

7.1 Downloading an Object

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can use the **get_object** function to download an object.

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
object_info	obs_object_info *	Mandatory	Object name and version number. For non-multi-version objects, set **version** to **0**.
get_conditions	obs_get_conditions *	Mandatory	Sets filter conditions of the object and read range.
encryption_params	server_side_encryption_params *	Optional	Obtains the decryption settings of an object.

Field	Type	Mandatory or Optional	Description
handler	obs_get_object_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample Code

```
static void test_get_object()
{
    char *file_name = "./test";
    obs_object_info object_info;
    // Initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // Set the object to be downloaded.
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key = "<object key>";
    object_info.version_id = "<object version ID>";
    // Set the structure for storing downloaded object data based on service requirements.
    get_object_callback_data data;
    data.ret_status = OBS_STATUS_BUTT;
    data.outfile = write_to_file(file_name);
    // Define range download parameters.
    obs_get_conditions getcondition;
    memset(&getcondition, 0, sizeof(obs_get_conditions));
    init_get_properties(&getcondition);
    getcondition.start_byte = "<start byte>";
    // Download length. The default value is 0, indicating that the object end is read.
    getcondition.byte_count = "<byte count>";

    // Customize callback function for download.
    obs_get_object_handler get_object_handler =
    {
        { &response_properties_callback,
          &get_object_complete_callback},
        &get_object_data_callback
    };

    get_object(&option, &object_info, &getcondition, 0, &get_object_handler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("get object successfully. \n");
    }
    else
    {
        printf("get object faied(%s).\n", obs_get_status_name(data.ret_status));
```



```
    }
    fclose(data.outfile);
}
```

NOTE

- You can perform operations on the input streams of an object to read and write the object contents to a local file or to the memory.
- **getcondition** can be left empty, indicating that the entire object is downloaded.

7.2 Performing a Conditioned Download

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

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

You can set the following conditions:

Parameter	Description
obs_get_conditions.if_modified_since	Returns the object if it has been modified since the specified time; otherwise, an error is returned.
obs_get_conditions.if_not_modified_since	Returns the object if it has not been modified since the specified time; otherwise, an error is returned.
obs_get_conditions.if_match_etag	Returns the source object if its ETag is the same as the one specified by this parameter; otherwise, an exception is thrown.
obs_get_conditions.if_not_match_etag	Returns the source object if its ETag is different from the one specified by this parameter; otherwise, an exception is thrown.

NOTE

- The ETag of an object is the MD5 check value of the object.
- If the download request includes **if_not_modified_since** or **If-Match** and the specified condition is not met, an exception will be thrown with HTTP status code **412 precondition failed** returned.
- If a request includes **if_modified_since** or **If-None-Match**, and the specified condition is not met, **304 Not Modified** will be returned.

Sample code:

```
static void test_get_object_by_range()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Range download parameters
    obs_get_conditions getcondition;
    memset(&getcondition, 0, sizeof(obs_get_conditions));
    init_get_properties(&getcondition);
    getcondition.if_match_etag = "<object etag>";
    getcondition.if_modified_since = "<time object modified>";
    getcondition.if_not_match_etag = "<not matched etag>";
    getcondition.if_not_modified_since = "<time object modified>";
    // Download object information.
    obs_object_info object_info;
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key = "<object key>";
    object_info.version_id = "<object version ID>";
    // Local file information after download
    get_object_callback_data data;
    data.ret_status = OBS_STATUS_BUTT;
    data.outfile = write_to_file("<file path>");
     // Set response callback function.
    obs_get_object_handler getobjectHandler =
    {
        { &response_properties_callback, &get_object_complete_callback},
        &get_object_data_callback
    };

    // Download objects.
    get_object(&option,&object_info,&getcondition,0,&getobjectHandler,&data);
    fclose(data.outfile);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("get object successfully. \n");
    }
    else
    {
        printf("get object faied(%s).\n", obs_get_status_name(data.ret_status));
    }
}
```

7.3 Downloading an Archive Object

NOTICE

If you have any questions during the development, post them on the **Issues** page of GitHub.

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

Option	Description	Value in OBS C SDK
Expedited restore	Data can be restored within 1 to 5 minutes.	OBS_TIER_EXPEDITED
Standard restore	Data can be restored within 3 to 5 hours. This is the default option.	OBS_TIER_STANDARD

CAUTION

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

You can call **restore_object** to restore an Archive object. Sample code:

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
object_info	obs_object_info *	Mandatory	Object name and version number. For non-multi-version objects, set **version** to **0**.
days	char *	Mandatory	Retention period of the restored object
tier	obs_tier	Optional	Restore options: **obs_tier.OBS_TIER_EXPEDITED** or **obs_tier.OBS_TIER_STANDARD**
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data


```
static void test_restore_object()
{
    // Define object information.
    obs_object_info object_info;
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key = "<object key>";
    object_info.version_id = "<object version ID>";
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_response_handler handler =
    {
      &response_properties_callback, &response_complete_callback
    };
    // Restore the object.
    obs_tier tier = OBS_TIER_EXPEDITED;
    restore_object(&option, &object_info, "<stored time>",tier,&handler, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {
        printf("restore object successfully. \n");
    }
    else
    {
        printf("restore object faied(%s).\n", obs_get_status_name(ret_status));
        return;
    }
    // Download object callback data.
    get_object_callback_data data;
    data.ret_status = OBS_STATUS_BUTT;
    data.outfile = write_to_file("<file path>");
    // Set response callback function.
    obs_get_object_handler getobjectHandler =
    {
        { &response_properties_callback, &get_object_complete_callback},
        &get_object_data_callback
    };
    // Download an object.
    get_object(&option,&object_info, 0,0,&getobjectHandler,&data);
    fclose(data.outfile);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("get object successfully. \n");
    }
    else
    {
        printf("get object faied(%s).\n", obs_get_status_name(data.ret_status));
    }
}
```

NOTE

- The object specified in **object_info.key** must be in the OBS Archive storage class. Otherwise, an error will be reported when you call this API.
- <*Retention period of the restored object*> indicates how long (1 to 30 days) the restored object will be retained.

7.4 Performing a Resumable Download

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

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

You can use **download_file** to perform a resumable download. The following table describes the parameters involved in this API.

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
key	char *	Mandatory	Object name
version_id	char *	Mandatory	Object version ID
download_file_config	obs_download_file_configuration *	Mandatory	For details about how to download files, see the **obs_download_file_configuration** member description in the following table.

Field	Type	Mandatory or Optional	Description
get_conditions	obs_get_conditions *	Mandatory	For details about how to set the object filter conditions and read range, see **Performing a Conditioned Download**.
encryption_params	server_side_encryption_params *	Optional	Encryption setting of the uploaded object
handler	obs_download_file_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

The following table describes the structure of **obs_download_file_configuration**.

Field	Type	Mandatory or Optional	Description
downLoad_file	char *	Mandatory	Local path to which the object is downloaded. If the value is null, the downloaded object is saved in the directory where the program is executed.
part_size	uint64_t	Mandatory	Part size, in bytes. The value ranges from 5 MB (default) to 5 GB.
task_num	int	Mandatory	Maximum number of parts that can be concurrently downloaded. The default value is **1**.
enable_check_point	int	Mandatory	Whether to enable the resumable upload mode. The default value is **0**, which indicates that this mode is disabled.

Field	Type	Mandatory or Optional	Description
check_point_file	char *	Mandatory	File used to record the download progress. This parameter is effective only in the resumable download mode. If the value is null, the file is in the same local directory as the downloaded object.

Sample code:

```
void downloadFileResultCallback(obs_status status,
                                char *resultMsg,
                                int partCountReturn,
                                obs_download_file_part_info * downloadInfoList,
                                void *callbackData);
//Declare the callback function.
static void test_download_file(char *filename, char *key)
{
    obs_status ret_status = OBS_STATUS_BUTT;
    uint64_t uploadSliceSize =5L * 1024 * 1024;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Initialize getConditions.
    obs_get_conditions getConditions;
    memset_s(&getConditions,sizeof(obs_get_conditions),0,sizeof(obs_get_conditions));
    init_get_properties(&getConditions);
    // Resumable download object information
    obs_download_file_configuration downloadFileConfig;
    memset_s(&downloadFileConfig,sizeof(obs_download_file_configuration),0,
                sizeof(obs_download_file_configuration));
    downloadFileConfig.check_point_file = NULL;
    downloadFileConfig.enable_check_point = 1;
    downloadFileConfig.part_size = uploadSliceSize;
    downloadFileConfig.task_num = 10;
    downloadFileConfig.downLoad_file= filename;
    // Set response callback function.
    obs_download_file_response_handler Handler =
    {
        {&response_properties_callback, &response_complete_callback_for_multi_task},
        &downloadFileResultCallback
    };
    initialize_break_point_lock();
    download_file(&option, key, 0,&getConditions,0,&downloadFileConfig,
            &Handler, &ret_status);
```



```
    deinitialize_break_point_lock();
    if (OBS_STATUS_OK == ret_status) {
        printf("test download file successfully. \n");
    }
    else
    {
        printf("test download file faied(%s).\n", obs_get_status_name(ret_status));
    }
}
//downloadFileResultCallback is used as an example here. The printf statements can be replaced with custom logging print statements.
void downloadFileResultCallback(obs_status status,
                                char *resultMsg,
                                int partCountReturn,
                                obs_download_file_part_info * downloadInfoList,
                                void *callbackData)
{
    int i=0;
    obs_download_file_part_info * pstDownloadInfoList = downloadInfoList;
    printf("status return is %d(%s)\n", status, obs_get_status_name(status));
    printf("%s",resultMsg);
    printf("partCount[%d]\n",partCountReturn);
    for(i=0;i<partCountReturn;i++)
    {
        printf("partNum[%d],startByte[%llu],partSize[%llu],status[%d]\n",
        pstDownloadInfoList->part_num,
        pstDownloadInfoList->start_byte,
        pstDownloadInfoList->part_size,
        pstDownloadInfoList->status_return);
        pstDownloadInfoList++;
    }
    if (callbackData) {
        obs_status* retStatus = (obs_status*)callbackData;
        (*retStatus) = status;
    }
}
```

NOTE

- The API for resumable download, which is implemented based on **Performing a Multipart Upload**, is an encapsulated and enhanced version of partial download.
- This API saves resources and improves efficiency upon the re-download, and speeds up the download process by concurrently downloading parts. Because this API is transparent to users, users are free from concerns about internal service details, such as the creation and deletion of checkpoint files, division of objects, and concurrent download of parts.
- The default value of the **enable_check_point** parameter is **0**, which indicates that the resumable download mode is disabled. In such cases, the API for resumable download degrades to the simple encapsulation of partial download, and no checkpoint file will be generated.
- **check_point_file** is valid only when **enable_check_point** is set to 1.
- Use the **task_num** parameter to configure how many concurrent tasks there can be in downloading a single object.

7.5 Processing an Image

NOTICE

If you have any questions during the development, post them on the **Issues** page of GitHub.

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

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

Sample code:

```
static void test_get_object_image()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Image transcoding configuration
    obs_get_conditions getcondition;
    memset(&getcondition, 0, sizeof(obs_get_conditions));
    init_get_properties(&getcondition);
    getcondition.image_process_config.image_process_mode = obs_image_process_cmd;
    getcondition.image_process_config.cmds_stylename = "resize,m_fixed,w_100,h_100/rotate,90";
    // Download object information.
    obs_object_info object_info;
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key = "<object key>";
    object_info.version_id = "<object version ID>";
    // Download object callback data.
    get_object_callback_data data;
    data.ret_status = OBS_STATUS_BUTT;
    data.outfile = write_to_file("<file path>");
    // Set response callback function.
    obs_get_object_handler getobjectHandler =
    {
        { &response_properties_callback, &get_object_complete_callback},
        &get_object_data_callback
    };

    // Download an object.
    get_object(&option,&object_info,&getcondition,0,&getobjectHandler,&data);
    fclose(data.outfile);
     if (OBS_STATUS_OK == data.ret_status) {
            printf("get object successfully. \n");
        }
    else
    {
        printf("get object faied(%s).\n", obs_get_status_name(data.ret_status));
    }
}
```

NOTE

- Use **getcondition.image_process_config** to specify the image processing parameters.
- Currently, image processing parameters can only be processed in the **image/commands** format.
- Image processing parameters can be processed in cascading mode. This indicates that multiple commands can be performed on an image in sequence.

8 Object Management

8.1 Obtaining Object Properties

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

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

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
object_info	obs_object_info *	Mandatory	Object name and version number. For non-multi-version objects, set **version** to **0**.
encryption_params	server_side_encryption_params *	Optional	Parameters related to server-side encryption
handler	obs_get_object_handler *	Mandatory	Callback function

Field	Type	Mandatory or Optional	Description
callback_data	void *	Optional	Callback data

Sample Code

Sample code:

```
static void test_get_object_metadata()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Object information
    obs_object_info object_info;
    memset(&object_info,0,sizeof(obs_object_info));
    object_info.key = "<object key>";
    object_info.version_id = "<object version ID>";
    // Set response callback function.
    obs_response_handler response_handler =
    {
        &response_properties_callback, &response_complete_callback
    };
    // Obtain object properties.
    get_object_metadata(&option,&object_info,0, &response_handler,&ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("get object metadata successfully. \n");
    }
    else
    {
        printf("get object metadata failed.\n", obs_get_status_name(ret_status));
    }
}
```

8.2 Managing Object ACLs

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

Object ACLs, similar to bucket ACLs, support predefined ACLs (For details, see **Managing Bucket ACLs**) and direct configurations.

An object **ACL** can be configured in any of the following ways:

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

Specifying a Pre-defined Access Control Policy During Object Upload

Sample code:

```
static void test_put_object_acl()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // Initialize put_properties which can be used to set object properties.
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    // Specify a pre-defined access control policy.
    put_properties.canned_acl = OBS_CANNED_ACL_PUBLIC_READ_WRITE;
    // Callback data
    put_file_object_callback_data data;
    memset(&data, 0, sizeof(put_file_object_callback_data));
    // Read the file to be uploaded to the callback data.
    data.infile = 0;
    data.noStatus = 1;
    content_length = read_bytes_from_file("<Uploaded filename>", &data);
    // Callback function
    obs_put_object_handler putobjectHandler =
    {
        { &response_properties_callback, &response_complete_callback },
        &put_buffer_object_data_callback
     };
    // Upload data streams.
    put_object(&option,"<object key>", content_length, &put_properties, 0, &putobjectHandler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("put object from file successfully. \n");
    }
    else
    {
        printf("put object failed(%s).\n",
               obs_get_status_name(data.ret_status));
    }
}
```

Setting a Pre-defined Access Control Policy for an Object

You can use **set_object_acl_by_head** to set object properties. The following table describes the parameters.

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
object_info	obs_object_info *	Mandatory	Object name and version number. For non-multi-version objects, set **version** to **0**.
canned_acl	obs_canned_acl	Mandatory	For details about how to manage bucket access rights, see **Table 5-1** in **Managing Bucket ACLs**.
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample code:

```
void test_set_object_acl_byhead()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    obs_canned_acl canned_acl = OBS_CANNED_ACL_PUBLIC_READ_WRITE;
    obs_object_info object_info;
    object_info.key = "<object key>";
    object_info.version_id = "<object version ID>";
    //Set a pre-defined access control policy for an object.
    set_object_acl_by_head(&option, &object_info, canned_acl, &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("set bucket acl by head successfully. \n");
    }
```



```
    else
    {
        printf("set bucket acl by head failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

Directly Setting an Object ACL

You can use **set_object_acl** to set object properties. The following table describes the parameters.

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
aclinfo	manager_acl_info *	Mandatory	Structure for managing ACL permission information
aclinfo->object_info	obs_object_info *	Mandatory	Object name and version number. For non-multi-version objects, set **version** to **0**.
aclinfo->owner_id	char *	Optional	User account ID
aclinfo->acl_grant_count_return	int *	Mandatory	Pointer to the number of returned **aclinfo->acl_grants**
aclinfo->acl_grants	obs_acl_grant *	Mandatory	Pointer to the permission information structure. For details, see **Table 5-2** in **Managing Bucket ACLs**.
aclinfo->object_delivered	obs_object_delivered	Optional	Indicates whether the object ACL inherits the ACL of the bucket. Valid values are **OBJECT_DELIVERED_TRUE** and **OBJECT_DELIVERED_FALSE**. The default value is **OBJECT_DELIVERED_TRUE**.
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample code:

```
void test_set_object_acl()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // Set response callback function.
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // Define the object ACL
    manager_acl_info aclinfo;
    init_acl_info(&aclinfo);
    aclinfo.object_info.key = "<object key>";
    aclinfo.object_info.version_id = "<object version ID>";
    // Set the object ACL.
    set_object_acl(&option, &aclinfo, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("set object acl successfully. \n");
    }
    else
    {
        printf("set object acl failed(%s).\n", obs_get_status_name(ret_status));
    }
    // Destroy the memory
    deinitialize_acl_info(&aclinfo);
}
```

NOTE

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

Obtaining the Object ACL

You can call **get_object_acl** to obtain an object ACL. Sample code:

```
void test_get_object_acl()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
```

```
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    manager_acl_info *aclinfo = malloc_acl_info();
    aclinfo->object_info.key = "<object key>";
    aclinfo->object_info.version_id = "<object version ID>";
    // Obtain the object ACL.
    get_object_acl(&option, aclinfo, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {
        printf("get object acl: ----------");
        printf("%s %d %s %s\n", aclinfo->owner_id, aclinfo->object_delivered,
            aclinfo->object_info.key, aclinfo->object_info.version_id);
        if (aclinfo->acl_grant_count_return)
        {
            print_grant_info(*aclinfo->acl_grant_count_return, aclinfo->acl_grants);
        }
    }
    else
    {
        printf("get object acl failed(%s).\n", obs_get_status_name(ret_status));
    }
    // Destroy the memory
    free_acl_info(&aclinfo);
}
```

8.3 Listing Objects

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can call **list_bucket_objects** to list objects in a bucket.

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
prefix	char *	Optional	Name prefix that the objects to be listed must contain

Field	Type	Mandatory or Optional	Description
marker	char *	Optional	Object name to start with when listing objects in a bucket. All objects are listed in the lexicographical order
delimiter	char *	Optional	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.
maxkeys	int	Mandatory	Maximum number of objects listed in the response body. The value ranges from 1 to 1000. If the value is not in this range, 1000 objects are returned by default.
handler	obs_list_objects_handler *	Mandatory	Callback function

Field	Type	Mandatory or Optional	Description
callback_data	void *	Optional	Callback data

Sample Code

```
static void test_list_bucket_objects(char *bucket_name)
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // Set response callback function.
    obs_list_objects_handler list_bucket_objects_handler =
    {
        { &response_properties_callback, &listobjects_complete_callback },
        &list_objects_callback
    };

    // Customize callback data.
    list_bucket_callback_data data;
    memset(&data, 0, sizeof(list_bucket_callback_data));
    // List objects.
    list_bucket_objects(&option, "<prefix>", "<marker>", "<delimiter>", "<maxkeys>", &list_bucket_objects_handler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("list bucket objects successfully. \n");
    }
    else
    {
        printf("list bucket objects failed(%s).\n",
        obs_get_status_name(data.ret_status));
    }
}
```

NOTE

- Information of a maximum of 1000 objects can be listed each time. If a bucket contains more than 1000 objects and **list_objects_data.is_truncated** is **true** in the returned result, not all objects are returned. In such cases, you can use **list_objects_data.next_marker** to obtain the start position for next listing.
- To obtain all objects, you can list them in paging mode.

8.4 Deleting Objects

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

NOTE

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

Deleting a Single Object

You can use **delete_object** to delete a single object. The parameters are described as follows:

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
object_info	obs_object_info *	Mandatory	Object name and version number. For non-multi-version objects, set **version** to **0**.
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample code:

```
static void test_delete_object(char *key, char *version_id, char *bucket_name)
{
    obs_status ret_status = OBS_STATUS_BUTT;

    // Create and initialize object information.
    obs_object_info object_info;
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key = key;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
```



```
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // Set response callback function.
    obs_response_handler resqonseHandler =
    {
        &response_properties_callback,
        &response_complete_callback
    };

    // Delete an object.
    delete_object(&option,&object_info,&resqonseHandler, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {
        printf("delete object successfully. \n");
    }
    else
    {
        printf("delete object failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

Batch Deleting Objects

You can call **batch_delete_objects** to delete multiple objects in batches.

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.

The parameters are as follows:

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
object_info	obs_object_info *	Mandatory	Name and version number of the object to be deleted. For non-multi-version objects, set **version** to **0**.

Field	Type	Mandatory or Optional	Description
delobj	obs_delete_object_info*	Mandatory	The number of objects to be deleted. Specifies the quiet mode.
put_properties	obs_put_properties*	Optional	Sets the verification properties of the object to be deleted.
handler	obs_delete_object_handler*	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample code:

```
static void test_batch_delete_objects()
{
    obs_status ret_status = OBS_STATUS_BUTT;

    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);

    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // Initialize information of the objects to be deleted.
    obs_object_info objectinfo[100];
    objectinfo[0].key = "obj1";
    objectinfo[0].version_id = "versionid1";
    objectinfo[1].key = "obj2";
    objectinfo[1].version_id = "versionid2";

    obs_delete_object_info delobj;
    memset_s(&delobj,sizeof(obs_delete_object_info),0,sizeof(obs_delete_object_info));
    delobj.keys_number = 2;

    // Set response callback function.
    obs_delete_object_handler handler =
    {
        {&response_properties_callback, &response_complete_callback},
        &delete_objects_data_callback
    };

    // Delete objects in batches.
    batch_delete_objects(&option, objectinfo, &delobj, 0, &handler, &ret_status);
```



```
    if (OBS_STATUS_OK == ret_status) {
        printf("test batch_delete_objects successfully. \n");
    }
    else
    {
        printf("test batch_delete_objects faied(%s).\n", obs_get_status_name(ret_status));
    }
}
```

8.5 Copying an Object

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

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

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

NOTE

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

Copying an Object in Simple Mode

The parameters are as follows:

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
key	char *	Mandatory	Object name
version_id	char *	Optional	Object version ID
object_info	obs_copy_destination_object_info *	Mandatory	Indicates the information of the copied object.
object_info->destination_bucket	char *	Mandatory	Bucket where the target object is located

Field	Type	Mandatory or Optional	Description
object_info->destination_key	char *	Mandatory	Target object name
object_info->last_modified_return	int64_t *	Mandatory	Latest time when the object was modified
object_info->etag_return_size	int	Mandatory	Cache size of ETag
object_info->etag_return	char *	Mandatory	Cache of ETag
is_copy	unsigned int	Mandatory	Indicates whether the metadata of the target object is copied from the source object or replaced with the metadata contained in the request.
put_properties	obs_put_properties*	Optional	Properties of the uploaded object
encryption_params	server_side_encryption_params *	Optional	Configuring server-side encryption
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample code:

```
static void test_copy_object()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    char eTag[OBS_COMMON_LEN_256] = {0};
    int64_t lastModified;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
```



```
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set the destination object information.
    obs_copy_destination_object_info objectinfo ={0};
    objectinfo.destination_bucket = target_bucket;
    objectinfo.destination_key = destinationKey;
    objectinfo.etag_return = eTag;
    objectinfo.etag_return_size = sizeof(eTag);
    objectinfo.last_modified_return = &lastModified;
    // Set response callback function.
    obs_response_handler responseHandler =
    {
        &response_properties_callback,
        &response_complete_callback
    };
    // Copy an object.
    copy_object(&option, key, version_id, &objectinfo, 1, NULL, NULL,&responseHandler,&ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("test_copy_object  successfully. \n");
    }
    else
    {
        printf("test_copy_object failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

Copying an Object by Specifying Conditions

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

You can set the following conditions:

Parameter	Description
if_modified_since	Copies the source object if it has been modified since the specified time; otherwise, an exception is thrown.
if_not_modified_since	Copies the source object if it has not been modified since the specified time; otherwise, an exception is thrown.
if_match_etag	Copies the source object if its ETag is the same as the one specified by this parameter; otherwise, an exception is thrown.
if_not_match_etag	Copies the source object if its ETag is different from the one specified by this parameter; otherwise, an exception is thrown.

NOTE

- The ETag of the source object is the MD5 check value of the source object.
- If the object copy request includes **if_not_modified_since**, **if_match_etag**, **if_modified_since**, or **if_not_match_etag**, and the specified condition is not met, the copy will fail and an exception will be thrown with HTTP status code **412 precondition failed** returned.
- **if_modified_since** and **if_not_match_etag** can be used together. So do **if_not_modified_since** and **if_match_etag**.

Sample code:

```
static void test_copy_object_condition()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    char eTag[OBS_COMMON_LEN_256] = {0};
    int64_t lastModified;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // Set the destination object information.
    obs_copy_destination_object_info objectinfo ={0};
    objectinfo.destination_bucket = target_bucket;
    objectinfo.destination_key = destinationKey;
    objectinfo.etag_return = eTag;
    objectinfo.etag_return_size = sizeof(eTag);
    objectinfo.last_modified_return = &lastModified;
    // Set conditions.
    obs_put_properties putProperties={0};
    init_put_properties(&putProperties);
    putProperties.get_conditions.if_match_etag = "<etag>";
    putProperties.get_conditions.if_modified_since = "<time>";
    putProperties.get_conditions.if_not_match_etag = "<etag>";
    putProperties.get_conditions.if_not_modified_since = "<time>";
    // Set response callback function.
    obs_response_handler responseHandler =
    {
        &response_properties_callback,
        &response_complete_callback
    };
    // Copy an object.
    copy_object(&option, key, version_id, &objectinfo, 1, &putProperties,
NULL,&responseHandler,&ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("test_copy_object  successfully. \n");
    }
    else
    {
        printf("test_copy_object failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

Modifying an Object ACL

Sample code:


```
static void test_copy_object_acl()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    char eTag[OBS_COMMON_LEN_256] = {0};
    int64_t lastModified;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set the destination object information.
    obs_copy_destination_object_info objectinfo ={0};
    objectinfo.destination_bucket = target_bucket;
    objectinfo.destination_key = destinationKey;
    objectinfo.etag_return = eTag;
    objectinfo.etag_return_size = sizeof(eTag);
    objectinfo.last_modified_return = &lastModified;
    // Modify an object ACL.
    obs_put_properties putProperties={0};
    init_put_properties(&putProperties);
    putProperties.canned_acl = OBS_CANNED_ACL_PUBLIC_READ;
    // Set response callback function.
    obs_response_handler responseHandler =
    {
        &response_properties_callback,
        &response_complete_callback
    };
    // Copy an object.
    copy_object(&option, key, version_id, &objectinfo, 1, &putProperties,
NULL,&responseHandler,&ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("test_copy_object  successfully. \n");
    }
    else
    {
        printf("test_copy_object failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

8.6 Renaming an Object or Directory

NOTICE

If you have any questions during the development, post them on the **Issues** page of GitHub.

You can call **rename_object** to change the object name and directory name. This API applies only to parallel file systems (PFSs), instead of object buckets.

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	PFS parameters
key	char *	Mandatory	Name of the object or directory to be renamed
new_object_name	char *	Mandatory	New name
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample Code

The following code shows how to rename an object:

```
static void test_rename_object()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Name of the object or directory to be renamed
    char *key = "put_buffer_test";
    // New name
    char *new_key_name = "put_buffer_test_new";
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_response_handler response_handler =
    {
        &response_properties_callback, &response_complete_callback
    };
    // Rename an object.
    rename_object(&option, key, new_key_name, &response_handler,&ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("rename_object successfully. \n");
    }
    else
```



```
    {
        printf("rename_object failed.\n", obs_get_status_name(ret_status));
    }
}
```

NOTE

- **rename_object** applies only to buckets used as parallel file systems.
- If the object to be renamed does not exist, an error is reported (HTTP status code 404).

8.7 Truncating an Object

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can call **truncate_object** to truncate an object. This API applies only to buckets used as parallel file systems.

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
key	char *	Mandatory	Object name
object_length	uint64_t	Mandatory	Length to be truncated to
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample Code

The following code shows how to truncate an object:

```
static void test_truncate_object()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Object name
```



```
    char *key = "put_buffer_test";
    // Length of the object to be truncated to
    uint64_t object_length = 10240;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_response_handler response_handler =
    {
        &response_properties_callback, &response_complete_callback
    };
    // Truncate an object.
    truncate_object(&option, key, object_length, &response_handler,&ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("truncate_object successfully. \n");
    }
    else
    {
        printf("truncate_object failed.\n", obs_get_status_name(ret_status));
    }
}
```

NOTE

- **truncate_object** applies only to buckets used as parallel file systems.
- If the object to be truncated does not exist, an error is reported (HTTP status code 404).

9 Temporarily Authorized Request

9.1 What Is a Temporarily Authorized Request

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

A temporarily authorized request is a URL temporarily authorized by specifying the AK and SK, request method, and related parameters. This URL contains authentication information and therefore you can use this URL to perform the specific operation in OBS. When the URL is being generated, you need to specify the validity period for it. The URL for temporarily authorized request is generated by setting the **temp_auth_configure** structure.

The **temp_auth_configure** structure exists in the **obs_options** structure. This method is applicable to each C SDK API.

Parameter	Description	Structure in SDK
expires	Validity period of the generated temporary URL	obs_options. temp_auth_configure
temp_auth_callback	The callback function used to return the generated temporary URL	
callback_data	Callback data	

CAUTION

If a CORS or signature mismatch error occurs, refer to the following steps to troubleshoot the issue:

1. If CORS is not configured, you need to configure CORS rules on OBS Console. For details, see **Configuring CORS**.
2. If the signatures do not match, check whether signature parameters are correct by referring to **Authentication of Signature in a URL**. For example, during an object upload, the backend uses **Content-Type** to calculate the signature and generate an authorized URL, but if **Content-Type** is not set or is set to an incorrect value when the frontend uses the authorized URL, a CORS error occurs. To avoid this issue, ensure that **Content-Type** fields at both the frontend and backend are kept consistent.

9.2 Temporarily Authorized Request Example

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can use SDK APIs to transfer the **temp_auth_configure** structure parameter to generate a temporary authorized request URL. The sample codes in the following sessions create URLs for common operations, including bucket creation, as well as object upload, download, listing, and deletion.

NOTE

The URLs generated in the following examples are recorded in the **tmpAuthUrl.txt** file. You can use the generated URLs to perform corresponding operations.

9.2.1 URL for Creating a Bucket

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

```
static void test_create_bucket_auth()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/
```



```
intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    temp_auth_configure tempauth;

    tempAuthResult  ptrResult;
    memset_s(&ptrResult,sizeof(tempAuthResult),0,sizeof(tempAuthResult));
    // Callback data
    tempauth.callback_data = (void *)(&ptrResult);
    // Validity period
    tempauth.expires = 10;
    // Callback function returns and generates the temporary URL.
    tempauth.temp_auth_callback = &tempAuthCallBack_getResult;
    option.temp_auth = &tempauth;
    // Assign a value to the callback function.
    obs_response_handler response_handler =
    {
        &response_properties_callback,
        &response_complete_callback
    };
    // Call the API.
    create_bucket(&option, canned_acl, NULL, &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("the temporary signature url of create bucket generated successfully. The result is recorded in
the tmpAuthUrl.txt file. \n");
    }
    else
    {
        printf(" the temporary signature url of create bucket generation failed(%s).\n",
obs_get_status_name(ret_status));
    }
}
```

9.2.2 URL for Uploading an Object

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

```
static void test_put_object_auth()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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
ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/
intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    temp_auth_configure tempauth;

    tempAuthResult  ptrResult;
    memset_s(&ptrResult,sizeof(tempAuthResult),0,sizeof(tempAuthResult));
    // Callback data
    tempauth.callback_data = (void *)(&ptrResult);
    // Validity period
    tempauth.expires = 10;
    // Callback function returns and generates the temporary URL.
```

```
    tempauth.temp_auth_callback = &tempAuthCallBack_getResult;
    option.temp_auth = &tempauth;
    // Initialize the put_properties structure.
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    // Callback data
    put_file_object_callback_data data;
    memset(&data, 0, sizeof(put_file_object_callback_data));
    data.infile = 0;
    data.noStatus = 1;
    content_length = read_bytes_from_file(file_name, &data);
    // Set response callback function.
    obs_put_object_handler putobjectHandler =
    {
    { &response_properties_callback,
          &response_complete_callback },
        &put_buffer_object_data_callback
     };
    // Upload data streams.
    put_object(&option,key,content_length,&put_properties,0,&putobjectHandler,&data);
    if (OBS_STATUS_OK == data.ret_status) {printf("the temporary signature url of put object from file
generated successfully. The result is recorded in the tmpAuthUrl.txt file. \n");
    }
    else
    {
        printf("the temporary signature url of put object from file generation faied(%s).\n",
obs_get_status_name(data.ret_status));
    }
}
```

9.2.3 URL for Downloading an Object

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

```
static void test_get_object_auth()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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
ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/
intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    temp_auth_configure tempauth;

    tempAuthResult  ptrResult;
    memset_s(&ptrResult,sizeof(tempAuthResult),0,sizeof(tempAuthResult));
    // Callback data
    tempauth.callback_data = (void *)(&ptrResult);
    // Validity period
    tempauth.expires = 10;
    // Callback function returns and generates the temporary URL.
    tempauth.temp_auth_callback = &tempAuthCallBack_getResult;
    option.temp_auth = &tempauth;
```



```
    // Download object information.
    obs_object_info object_info;
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key =key;
    object_info.version_id = versionid;
    // Local file information after download
    char *file_name = local_file_name;
    get_object_callback_data data;
    data.ret_status = OBS_STATUS_BUTT;
    data.outfile = write_to_file(file_name);
    // Callback function
    obs_get_object_handler getobjectHandler =
    {
        { &response_properties_callback,
          &get_object_complete_callback},
        &get_object_data_callback
    };
    // Download an object.
    get_object(&option,&object_info,0,0,&getobjectHandler,&data);
    fclose(data.outfile);
    if (OBS_STATUS_OK == data.ret_status) {printf("the temporary signature url of get object generated
successfully. The result is recorded in the tmpAuthUrl.txt file. \n");
    }
    else
    {
        printf("the temporary signature url of get object generation faied(%s).\n",
obs_get_status_name(data.ret_status));
    }
}
```

9.2.4 URL for Listing Objects

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

```
static void test_list_object_auth()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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
ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/
intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    temp_auth_configure tempauth;

    tempAuthResult  ptrResult;
    memset_s(&ptrResult,sizeof(tempAuthResult),0,sizeof(tempAuthResult));
    // Callback data
    tempauth.callback_data = (void *)(&ptrResult);
    // Validity period
    tempauth.expires = 10;
    // Callback function returns and generates the temporary URL.
    tempauth.temp_auth_callback = &tempAuthCallBack_getResult;
    option.temp_auth = &tempauth;
    // The maximum number of objects to be listed
```



```
    int maxkeys  =  100;
    obs_list_objects_handler list_bucket_objects_handler =
    {
          { &response_properties_callback, &list_object_complete_callback },
          &list_objects_callback
     };
    list_object_callback_data data;
    memset(&data, 0, sizeof(list_object_callback_data));
    list_bucket_objects(&option, NULL, data.next_marker,NULL, maxkeys,
    &list_bucket_objects_handler, &data);
    if (OBS_STATUS_OK == data.ret_status) {printf("the temporary signature url of list bucket objects
generated successfully. The result is recorded in the tmpAuthUrl.txt file. \n");
    }
    else
    {
        printf("the temporary signature url of list bucket objects generation failed(%s).\n",
                          obs_get_status_name(data.ret_status));
    }
}
```

9.2.5 URL for Deleting an Object

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

```
static void test_delete_object_auth()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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
ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/
intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    temp_auth_configure tempauth;

    tempAuthResult  ptrResult;
    memset_s(&ptrResult,sizeof(tempAuthResult),0,sizeof(tempAuthResult));
    // Callback data
    tempauth.callback_data = (void *)(&ptrResult);
    // Validity period
    tempauth.expires = 10;
    // Callback function returns and generates the temporary URL.
    tempauth.temp_auth_callback = &tempAuthCallBack_getResult;
    option.temp_auth = &tempauth;
    // Object information
    obs_object_info object_info;
    memset(&object_info,0,sizeof(obs_object_info));
    object_info.key = key;
    // Assign a value to the callback function.
    obs_response_handler resqonseHandler =
    {
    &response_properties_callback,
       &response_complete_callback
    };
```



```
    // Delete an object.
    delete_object(&option,&object_info,&resqonseHandler, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {printf("the temporary signature url of delete object generated successfully. The result is recorded in the
tmpAuthUrl.txt file. \n");
    }
    else
    {
        printf("the temporary signature url of delete object generation failed(%s).\n",
obs_get_status_name(ret_status));
    }
}
```


10 Accessing OBS Through a User-Defined Domain Name

10.1 Configuring a User-Defined Domain Name

To access OBS through a user-defined domain name, you need to first configure a domain name on the Console.

Configuring a User-Defined Domain Name

To enable access to an OBS bucket through a user-defined domain name, you need to first associate the user-defined domain name with the domain name of the bucket. For details, see **User-Defined Domain Name Introduction** and **User-Defined Domain Name Configuration**.

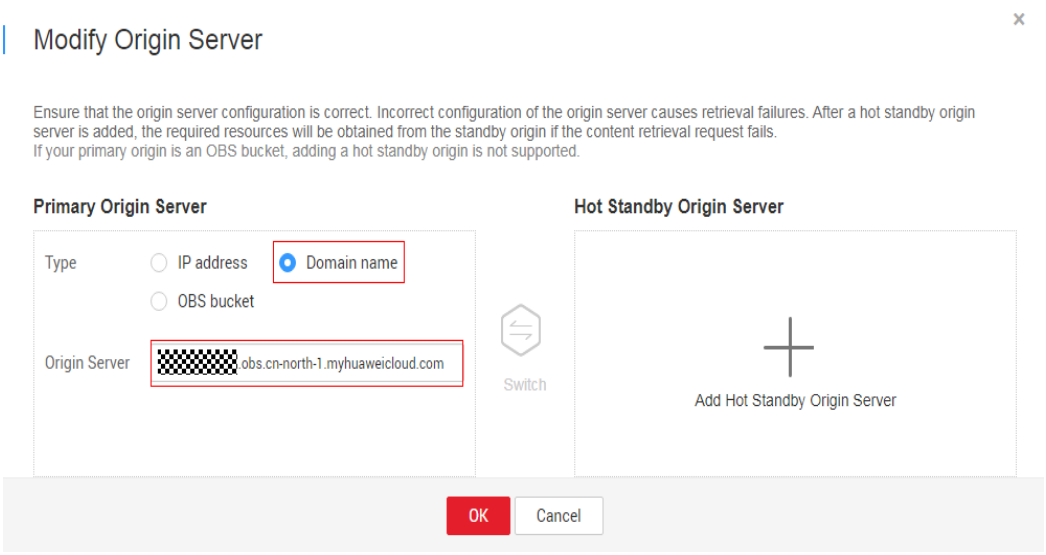
CAUTION

If a user-defined domain name is configured as an acceleration domain name with CDN, you need to enable it to access OBS by configuring settings on CDN console.

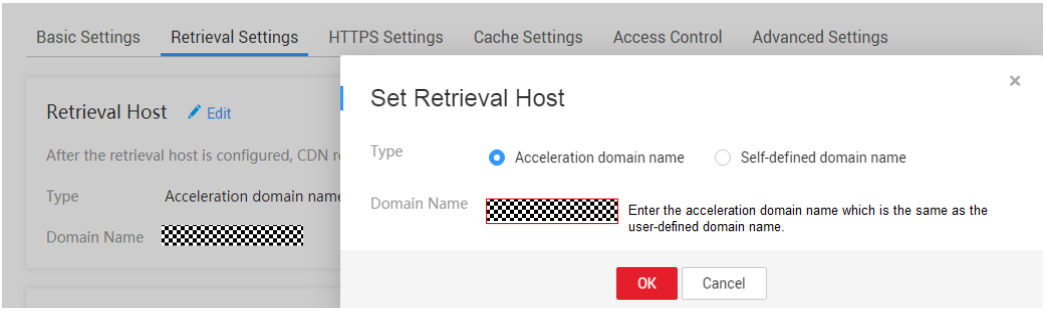
The example shows how to configure the user-defined domain name with CDN on Huawei Cloud.

Step 1 Log in to the Huawei Cloud CDN console and select **Domains** from the navigation pane on the left. You can view all configured CDN domain names.

Step 2 Configure the origin server. Click the user-defined domain name to be used to go to the configuration page. Modify the origin server. Set **Type** of **Primary Origin Server** to **Domain name**, and set **Origin Server** to the domain name of the OBS bucket to be accessed.



Step 3 Configure the retrieval host. The retrieval host must be specified as the acceleration domain name, that is, your user-defined domain name. Otherwise, the retrieval authentication may fail.



----End

10.2 Using a User-Defined Domain Name to Access OBS

When using a user-defined domain name to access OBS, you can call APIs as you normally do, except that you need to configure **option** as shown below:

```
option.bucket_options.useCname = true;
option.bucket_options.host_name = "yourCustomDomain";
```

Using a user-defined domain name to upload an object:

```
static void test_put_object_from_file()
{
    // Name of the object
    char *key = "put_file_test";
    // File to be uploaded
    char file_name[256] = "./put_file_test.txt";
    uint64_t content_length = 0;
    // Initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
```



```
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    option.bucket_options.certificate_info = "<Your certificate>";

    // Specify a user-defined domain name.
    option.bucket_options.useCname = true;
    option.bucket_options.host_name = "yourCustomDomain";
    // Initialize the properties of the object.
    obs_put_properties put_properties;
    init_put_properties(&put_properties);

    // Initialize the structure for storing data to be uploaded.
    put_file_object_callback_data data;
    memset(&data, 0, sizeof(put_file_object_callback_data));
    // Open the file and obtain the file length.
    content_length = open_file_and_get_length(file_name, &data);
    // Set the callback function.
    obs_put_object_handler putobjectHandler =
    {
        { &response_properties_callback, &put_file_complete_callback },
        &put_file_data_callback
    };

    put_object(&option, key, content_length, &put_properties, 0, &putobjectHandler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("put object from file successfully. \n");
    }
    else
    {
        printf("put object failed(%s).\n",
               obs_get_status_name(data.ret_status));
    }
}
```

Using a user-defined domain name to download an object:

```
static void test_get_object()
{
    char *file_name = "./test";
    obs_object_info object_info;
    // Initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // Specify a user-defined domain name.
    option.bucket_options.useCname = true;
    option.bucket_options.host_name = "yourCustomDomain";

    // Specify the object to be downloaded.
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key = "<object key>";
```

```
    object_info.version_id = "<object version ID>";
    //Set the structure for storing downloaded data based on service requirements.
    get_object_callback_data data;
    data.ret_status = OBS_STATUS_BUTT;
    data.outfile = write_to_file(file_name);
    // Define a range to download.
    obs_get_conditions getcondition;
    memset(&getcondition, 0, sizeof(obs_get_conditions));
    init_get_properties(&getcondition);
    getcondition.start_byte = "<start byte>";
    // Define a size to download. The default value is 0, indicating that the object is read to the end.
    getcondition.byte_count = "<byte count>";

    // Define a callback function.
    obs_get_object_handler get_object_handler =
    {
        { &response_properties_callback,
          &get_object_complete_callback},
        &get_object_data_callback
    };

    get_object(&option, &object_info, &getcondition, 0, &get_object_handler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("get object successfully. \n");
    }
    else
    {
        printf("get object faied(%s).\n", obs_get_status_name(data.ret_status));
    }
    fclose(data.outfile);
}
```


11 Versioning Management

11.1 Versioning Overview

NOTICE

If you have any questions during the development, post them on the **Issues** page of GitHub.

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

For details, see **Versioning**.

11.2 Setting Versioning Status for a Bucket

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

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

Versioning Status	Description	Value in OBS C SDK
Enabled	1. OBS creates a unique version ID for each uploaded object. Namesake objects are not overwritten and are distinguished by their own version IDs. 2. Objects can be downloaded by specifying the version ID. By default, the latest object 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 **list_bucket_objects** is called. You can call **list_versions** 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.	OBS_VERSION_STATUS_ENABLED

Versioning Status	Description	Value in OBS C SDK
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 latest object is downloaded if no version ID is specified. 4. Objects can be deleted by version ID. If an object is deleted with no version ID specified, the object is only attached with a deletion mark and version ID **null**. Objects with version ID **null** are physically deleted. 5. Except for delete markers, storage space occupied by objects with all version IDs is billed.	OBS_VERSION_STATUS_SUSPENDED

The following table describes the parameters involved in this API.

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
version_status	char *	Mandatory	Indicates the multi-version status of a bucket: **OBS_VERSION_STATUS_ENABLED**, **OBS_VERSION_STATUS_SUSPENDED**.

Field	Type	Mandatory or Optional	Description
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

```
static void test_set_bucket_version()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
     // Set response callback function.
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // Enable bucket versioning.
    set_bucket_version_configuration(&option, OBS_VERSION_STATUS_ENABLED,
        &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("set bucket version successfully. \n");
    }
    else
    {
        printf("set bucket version failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

11.3 Viewing Versioning Status of a Bucket

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can call **get_bucket_version_configuration** to view the versioning status of a bucket. Sample code:

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
status_return_size	int	Mandatory	Size of the versioning status cache
status_return	char *	Mandatory	Versioning status cache
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

```
static void test_get_bucket_version()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
     // Set response callback function.
    obs_response_handler response_handler =
    {
        &response_properties_callback,
        &response_complete_callback
    };
    // Obtain bucket versioning status.
    char status[OBS_COMMON_LEN_256] = {0};
    get_bucket_version_configuration(&option, sizeof(status), status,
        &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("get bucket version successfully.\n policy=(%s)\n", status);
    }
    else
    {
        printf("get bucket version failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```


11.4 Obtaining a Versioning Object

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can call **get_object** to pass the version ID (**obs_object_info .version_id**) to obtain a versioning object. Sample code is as follows:

```
static void test_get_object_version()
{
    // Create and initialize the object, and specify the object version through obs_object_info.key.
    obs_object_info object_info;
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key =key;
    object_info.version_id = versionid;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Create and Initialize callback data.
    char *file_name = "./test";
    get_object_callback_data data;
    data.ret_status = OBS_STATUS_BUTT;
    data.outfile = NULL;
    data.outfile = write_to_file(file_name);
    // Set response callback function.
    obs_get_object_handler getobjectHandler =
    {
        { &response_properties_callback,
          &get_object_complete_callback},
        &get_object_data_callback
    };
    // Obtain a versioning object.
    get_object(&option,&object_info,0,0,&getobjectHandler,&data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("get object successfully. \n");
    }
    else
    {
        printf("get object faied(%s).\n", obs_get_status_name(data.ret_status));
    }
    fclose(data.outfile);
}
```

NOTE

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

11.5 Copying a Versioning Object

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

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

```
static void test_copy_object_version()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    char eTag[OBS_COMMON_LEN_256] = {0};
    int64_t lastModified;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set the destination object information.
    obs_copy_destination_object_info objectinfo ={0};
    objectinfo.destination_bucket = target_bucket;
    objectinfo.destination_key = destinationKey;
    objectinfo.etag_return = eTag;
    objectinfo.etag_return_size = sizeof(eTag);
    objectinfo.last_modified_return = &lastModified;
    // Set response callback function.
    obs_response_handler responseHandler =
    {
        &response_properties_callback,
        &response_complete_callback
    };
    // Copy an object.
    copy_object(&option, key, version_id, &objectinfo, 1, NULL, NULL,&responseHandler,&ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("test_copy_object  successfully. \n");
    }
    else
    {
        printf("test_copy_object failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```


11.6 Restoring a Specific Archive Object Version

NOTICE

If you have any questions during the development, post them on the **Issues** page of GitHub.

You can call **restore_object** to restore an Archive object version by specifying **obs_object_info .version_id**. Sample code is as follows:

```
static void test_restore_object_version()
{
    // Create and initialize object information.
    obs_object_info object_info;
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key =key;
    object_info.version_id = versionid;
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    obs_tier tier = OBS_TIER_EXPEDITED;
    // Set response callback function.
    obs_response_handler handler =
    {
      &response_properties_callback,
      &response_complete_callback
    };
    // Restore a specific Archive object version.
    restore_object(&option, &object_info, days,tier,&handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("restore object successfully. \n");
    }
    else
    {
        printf("restore object faied(%s).\n", obs_get_status_name(ret_status));
    }
}
```

CAUTION

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

11.7 Listing Versioning Objects

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can call **list_versions** to list versioning objects in a bucket.

The following table describes the parameters involved in this API.

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
prefix	char *	Optional	Name prefix that the objects to be listed must contain
key_marker	char *	Optional	Versioning object name to start with when listing versioning objects in a bucket. All versioning objects are listed in the lexicographical order.
delimiter	char *	Optional	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, **common_prefixes**. (If a prefix is specified in the request, the prefix must be removed from the object name.)

Field	Type	Mandatory or Optional	Description
version_id_marker	char *	Optional	Version ID to start with when listing objects in a bucket. All objects are listed in the lexicographical order by object name and version ID. This parameter must be used together with **key_marker**.
maxkeys	int	Mandatory	Maximum number of objects listed in the response body. The value ranges from 1 to 1000. If the value is not in this range, 1000 objects are returned by default.
handler	obs_list_versions_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

NOTE

- If the value of **version_id_marker** is not a version ID specified by **key_marker**, **version_id_marker** is ineffective.
- The returned result of **list_versions** includes the versioning objects and delete markers.

Sample code:

```
static void test_list_versions()
{
    char *prefix = "o";
    char *key_marker = "obj";
    char *delimiter = "/";
    int maxkeys  =  10;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_list_versions_handler list_versions_handler =
```



```
    {
        { &response_properties_callback, &list_versions_complete_callback },
        &listVersionsCallback
    };
    // Create and Initialize callback data.
    list_versions_callback_data data;
    char* version_id_marker = NULL;
    memset(&data, 0, sizeof(list_bucket_callback_data));
    data.ret_status = OBS_STATUS_BUTT;
    snprintf(data.next_key_marker, sizeof(data.next_key_marker), "%s", key_marker);
    if (version_id_marker)
    {
        snprintf(data.next_versionId_marker, sizeof(data.next_versionId_marker), "%s", version_id_marker);
    }
    data.keyCount = 0;
    data.allDetails = 1;
    data.is_truncated = 0;
    // When listing versioned objects, you can use pagination by specifying the object prefix using prefix and specifying the number using maxkeys. delimiter specifies the character used for grouping. To group by folder, set delimiter to a forward slash (/). Use key_marker to specify the start position for listing versioned objects and use version_id_marker to specify the version ID.
    list_versions(&option, prefix, key_marker, delimiter, maxkeys, version_id_marker,
                  &list_versions_handler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("list versions successfully. \n");
    }
    else
    {
        printf("list versions failed(%s).\n", obs_get_status_name(data.ret_status));
    }
}
```

NOTE

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

11.8 Setting or Obtaining a Versioning Object ACL

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

Directly Setting a Versioning Object ACL

You can call **set_object_acl** to input the version ID (**version_id**) to set the ACL for a versioning object. Sample code is as follows:

```
static void test_set_object_acl_version()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";
```

```
    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // Initialize the ACL information and specify the object name and version ID.
    manager_acl_info aclinfo;
    init_acl_info(&aclinfo);
    aclinfo.object_info.key = key;
    aclinfo.object_info.version_id = version_id;
    // Set the object ACL.
    set_object_acl(&option, &aclinfo, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("set object acl successfully. \n");
    }
    else
    {
        printf("set object acl failed(%s).\n", obs_get_status_name(ret_status));
    }
    // Free memory.
    deinitialize_acl_info(&aclinfo);
}
```

NOTE

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.

Obtaining a Versioning Object ACL

You can call **get_object_acl** to pass the version ID (**version_id**) to obtain the ACL for a versioning object. Sample code is as follows:

```
static void test_get_object_acl_version()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    //Read the AK/SK from environment variables.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // Create an ACL structure and specify the object name and version ID.
    manager_acl_info *aclinfo = malloc_acl_info();
    aclinfo->object_info.key = key;
    aclinfo->object_info.version_id = version_id;
    // Obtain the object ACL.
    get_object_acl(&option, aclinfo, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {
        printf("get object acl: ----------");
```

```
        printf("%s %s %s %s\n", aclinfo->owner_id, aclinfo->owner_display_name,
            aclinfo->object_info.key, aclinfo->object_info.version_id);
        if (aclinfo->acl_grant_count_return)
        {
            print_grant_info(*aclinfo->acl_grant_count_return, aclinfo->acl_grants);
        }
    }
    else
    {
        printf("get object acl failed(%s).\n", obs_get_status_name(ret_status));
    }
    // Free memory.
    free_acl_info(&aclinfo);
}
```

11.9 Deleting Versioning Objects

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

Deleting a Single Versioning Object

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

```
static void test_delete_object_version()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize object information.
    obs_object_info object_info;
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key = key;
    object_info.version_id = version_id;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_response_handler resqonseHandler =
    {
        &response_properties_callback,
        &response_complete_callback
    };
    // Delete an object.
    delete_object(&option,&object_info,&resqonseHandler, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {
        printf("delete object successfully. \n");
    }
    else
```



```
    {
        printf("delete object failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

Deleting Versioning Objects in a Batch

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

```
static void test_batch_delete_object_version()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";
    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // Initialize information of the objects to be deleted.
    obs_object_info objectinfo[100];
    objectinfo[0].key = "obj1";
    objectinfo[0].version_id = "versionid1";
    objectinfo[1].key = "obj2";
    objectinfo[1].version_id = "versionid2";
    obs_delete_object_info delobj;
    memset_s(&delobj,sizeof(obs_delete_object_info),0,sizeof(obs_delete_object_info));
    delobj.keys_number = 2;
    // Set response callback function.
    obs_delete_object_handler handler =
    {
        {&response_properties_callback, &response_complete_callback},
        &delete_objects_data_callback
    };
    // Delete objects in batches.
    batch_delete_objects(&option, objectinfo, &delobj, 0, &handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("test batch_delete_objects successfully. \n");
    }
    else
    {
        printf("test batch_delete_objects faied(%s).\n", obs_get_status_name(ret_status));
    }
}
```


12 Lifecycle Management

12.1 Lifecycle Management Overview

NOTICE

If you have any questions during the development, post them on the **Issues** page of GitHub.

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 contains:

- 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 time for a noncurrent object version, which is specified in the following mode:
  – How many days after the object becomes a noncurrent object version
- Expiration time of a noncurrent object version, which can be specified in the following mode:
  – How many days after the object becomes a noncurrent object version
- Identifier specifying whether the setting is effective

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

NOTE

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

12.2 Setting Lifecycle Rules

NOTICE

If you have any questions during the development, post them on the **Issues** page of GitHub.

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

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
bucket_lifecycle_conf	obs_lifecycle_conf *	Mandatory	For details about the bucket lifecycle configuration, see the following table.
blcc_number	unsigned int	Mandatory	Number of array members in the array **bucket_lifecycle_conf**.
handler	obs_response_handler*	Mandatory	Callback function
callback_data	void *	Optional	Callback data

The following table describes the bucket lifecycle configuration structure **obs_lifecycle_conf**.

Field	Type	Mandatory or Optional	Description
date	const char *	If there is no **days** and no **transition**, **noncurrent_version_days**, or **noncurrent_version_transition**, this parameter is mandatory.	Indicates the time when the object expiration rule of the latest version takes effect. The value must conform to the ISO8601 standards and must be at 00:00 (UTC time).
days	const char *	If there is no **date** and no **transition**, **noncurrent_version_days**, **noncurrent_version_transition** is available, this parameter is mandatory.	Number of days when the expiration rule takes effect after the object creation (only applicable to latest versions of objects).
id	const char *	Optional	Unique identifier of a rule. The value can contain a maximum of 255 characters.
prefix	const char *	Mandatory	Object name prefix identifying one or more objects to which the rule applies.
status	const char *	Mandatory	Indicates whether the rule is enabled.
noncurrent_version_days	const char *	Optional	Number of days when the specified expiration rule takes effect after the object becomes a noncurrent version (only applicable to an object's noncurrent version). Container for the expiration time of objects' noncurrent versions. If versioning is enabled or suspended for a bucket, you can set this parameter to delete noncurrent versions of objects that match the lifecycle rule (only applicable to the noncurrent versions of objects).

Field	Type	Mandatory or Optional	Description
transition_num	unsigned int	If **transition** is not empty, this parameter is mandatory.	Number of array members in the array **transition**.
transition	obs_lifecycle_transition *	If there is no **date**, **days**, **noncurrent_version_transition**, or **noncurrent_version_days**, this parameter is mandatory.	Indicates the transition time and the object storage class after transition (valid only for the latest object version).
transition->date	const char *	This parameter is mandatory if there is **transition** but no **transition.days**.	Indicates the time when the object transition rule of the latest version takes effect. The value must conform to the ISO8601 standards and must be at 00:00 (UTC time).
transition->days	const char *	This parameter is mandatory if there is **transition** but no **transition.date**.	Number of days when the transition rule takes effect after the object creation (only applicable to latest versions of objects).
transition->storage_class	obs_storage_class	If there is **transition**, this parameter is mandatory.	The storage class to which the latest version object is modified.
noncurrent_version_transition_num	unsigned int	If **obs_lifecycle_noncurrent_transition** is not left blank, this parameter is mandatory.	Number of array members in the array **noncurrent_version_transition**.
noncurrent_version_transition	obs_lifecycle_noncurrent_transition *	If there is no **date**, **days**, **transition**, or **noncurrent_version_days**, this parameter is mandatory.	Transition time of noncurrent object versions and the object storage class after transition.

Field	Type	Mandatory or Optional	Description
noncurrent_version_transition->noncurrent_version_days	const char *	If there is **noncurrent_version_transition**, this parameter is mandatory.	Number of days when the specified transition rule takes effect after the object becomes a noncurrent version (only applicable to an object's noncurrent version).
noncurrent_version_transition->storage_class	obs_storage_class	If there is **noncurrent_version_transition**, this parameter is mandatory.	The storage class to which the noncurrent version object is modified.

Setting an Object Transition Policy

Sample code:

```
static void test_set_bucket_lifecycle_configuration1()
{
    obs_status  ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set the completed callback function.
    obs_response_handler response_handler =
    {
        NULL, &response_complete_callback
    };
    obs_lifecycle_conf bucket_lifecycle_conf;
    memset(&bucket_lifecycle_conf, 0, sizeof(obs_lifecycle_conf));
    // ID of the lifecycle rule
    bucket_lifecycle_conf.id = "test3";
    // Designate prefix "test".
    bucket_lifecycle_conf.prefix = "bcd";
    // The lifecycle rule takes effect.
    bucket_lifecycle_conf.status = "Enabled";
    // Specify that objects whose names contain the prefix will expire 10 days after creation.
    bucket_lifecycle_conf.days = "10";
    obs_lifecycle_transition transition;
    memset(&transition, 0, sizeof(obs_lifecycle_transition));
    // Specify that objects whose names contain the specified prefix will be transitioned 30 days after creation.
    transition.days = "30";
    // Specify the storage class that the object will be transitioned to.
    transition.storage_class = OBS_STORAGE_CLASS_STANDARD_IA;
    bucket_lifecycle_conf.transition = &transition;
    bucket_lifecycle_conf.transition_num = 1;
```



```
    obs_lifecycle_noncurrent_transition noncurrent_transition;
    memset(&noncurrent_transition, 0, sizeof(obs_lifecycle_noncurrent_transition));
    // Specify that objects whose names contain the specified prefix will be transitioned 30 days after
becoming noncurrent versions.
    noncurrent_transition.noncurrent_version_days = "30";
    // Specify the storage class of objects whose names contain the prefix after changing into noncurrent
versions.
    noncurrent_transition.storage_class = OBS_STORAGE_CLASS_STANDARD_IA;
    bucket_lifecycle_conf. noncurrent_version_transition = &noncurrent_transition;
    bucket_lifecycle_conf.noncurrent_version_transition_num = 1;
    set_bucket_lifecycle_configuration(&option, &bucket_lifecycle_conf, 1,
                &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("set bucket lifecycle configuration success.\n");
    }
    else
    {
        printf("set bucket lifecycle configuration failed(%s).\n",
                obs_get_status_name(ret_status));
    }
}
```

Setting an Object Expiration Time

Sample code:

```
static void test_set_bucket_lifecycle_configuration2()
{
    obs_options option;
    obs_status  ret_status = OBS_STATUS_BUTT;
    // Set option.
    init_obs_options(&option);
    option.bucket_options.host_name = HOST_NAME;
    option.bucket_options.bucket_name = bucket_name;

    //Read the AK/SK from environment variables.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set the completed callback function.
    obs_response_handler response_handler =
    {
        NULL, &response_complete_callback
    };
    obs_lifecycle_conf bucket_lifecycle_conf;
    memset(&bucket_lifecycle_conf, 0, sizeof(obs_lifecycle_conf));
    // ID of the lifecycle rule
    bucket_lifecycle_conf.id = "test1";
    // Designate prefix "test".
    bucket_lifecycle_conf.prefix = "test";
    // Specify that objects whose names contain the prefix will expire 10 days after creation.
    bucket_lifecycle_conf.days = "10";
    // Specify that objects whose names contain the specified prefix will expire 20 days after becoming
noncurrent versions.
    bucket_lifecycle_conf.noncurrent_version_days = "20";
    // The lifecycle rule takes effect.
    bucket_lifecycle_conf.status = "Enabled";
    set_bucket_lifecycle_configuration(&option, &bucket_lifecycle_conf, 1,
              &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
         printf("set bucket lifecycle configuration success.\n");
    }
    else
    {
        printf("set bucket lifecycle configuration failed(%s).\n",
               obs_get_status_name(ret_status));
    }
}
```


12.3 Viewing Lifecycle Rules

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can call **get_bucket_lifecycle_configuration()** to set lifecycle rules for a bucket. Sample code is as follows:

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
handler	obs_lifecycle_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample Code

```
static void test_get_bucket_lifecycle_configuration()
{
    obs_status  ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set callback function.
    obs_lifecycle_handler lifeCycleHandlerEx =
    {
        {&response_properties_callback, &response_complete_callback},
        &getBucketLifecycleConfigurationCallbackEx
    };
```



```
    get_bucket_lifecycle_configuration(&option, &lifeCycleHandlerEx, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("get_lifecycle_config success.\n");
    }
    else
    {
        printf("get_lifecycle_config faied(%s).\n", obs_get_status_name(ret_status));
    }
}
```

12.4 Deleting Lifecycle Rules

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can call **delete_bucket_lifecycle_configuration()** to delete lifecycle rules for a bucket. Sample code is as follows:

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample Code

```
static void test_delete_bucket_lifecycle_configuration()
{
    obs_status  ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/
```



```
intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set callback function.
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    delete_bucket_lifecycle_configuration(&option, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("test_delete_lifecycle_config success.\n");
    }
    else
    {
        printf("test_delete_lifecycle_config faied(%s).\n",
            obs_get_status_name(ret_status));
    }
}
```

13 Cross-Origin Resource Sharing (CORS)

13.1 CORS Overview

NOTICE

If you have any questions during the development, post them on the **Issues** page of GitHub.

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

For more information, see **CORS**.

13.2 Setting CORS Rules

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

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

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
obs_cors_conf_info	obs_bucket_cors_conf *	Mandatory	For details about CORS rules, see the following table.
conf_num	unsigned int	Mandatory	Number of array members in the array **obs_cors_conf_info**.
handler	obs_response_handler*	Mandatory	Callback function
callback_data	void *	Optional	Callback data

The following table describes the CORS rule structure **obs_bucket_cors_conf**.

Field	Type	Mandatory or Optional	Description
id	const char *	Optional	Object names in a bucket
allowed_method	const char **	Mandatory	Method allowed by a CORS rule
allowed_method_number	unsigned int	Mandatory	Number of **allowed_method**
allowed_origin	const char **	Mandatory	Indicates an origin that is allowed by a CORS rule. It is a character string and can contain a wildcard (*), and allows one wildcard character (*) at most.
allowed_origin_number	unsigned int	Mandatory	Number of **allowed_origin**.

Field	Type	Mandatory or Optional	Description
allowed_header	const char **	Optional	Indicates which headers are allowed in a PUT Bucket CORS request via the **Access-Control-Request-Headers**. If a request contains **Access-Control-Request- Headers**, only a CORS request that matches the configuration of **allowed_header** is considered as a valid request. Each **allowed_header** can contain at most one wildcard (*) and cannot contain spaces.
allowed_header_number	unsigned int	Optional	Number of **allowed_header**
max_age_seconds	const char *	Optional	Response time of CORS that can be cached by a client. It is expressed in seconds. Each CORS rule can contain only one **max_age_seconds**. It can be set to a negative value.
expose_header	const char **	Optional	Indicates a supplemented header in CORS responses. The header provides additional information for clients. It cannot contain spaces.
expose_header_number	unsigned int	Optional	Number of **expose_header**.

Sample Code

```
static void test_set_bucket_cors()
{
    obs_options option;
    obs_status  ret_status = OBS_STATUS_BUTT;
    obs_bucket_cors_conf bucketCorsConf;
    // Set option.
    init_obs_options(&option);
    option->bucket_options.hostName = "<your-endpoint>";
    option->bucket_options.bucketName = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option->bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option->bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set callback function.
    obs_response_handler response_handler =
    {
        NULL, &response_complete_callback
    };

    char *id_1= "1";
    // Specify the browser's cache time of the returned results of OPTIONS requests for specific resources, in seconds.
    char *max_age_seconds = "100";
    // Specify the request method, which can be GET, PUT, DELETE, POST, or HEAD.
    const char* allowedMethod_1[5] = {"GET","PUT","HEAD","POST","DELETE"};
    // Specify the origin of the cross-domain request.
    const char* allowedOrigin_1[2] = {"obs.example.com", "www.example.com"};
    // Specify response headers that users can access using application programs.
    const char* allowedHeader_1[2] = {"header-1", "header-2"};
    // Additional headers carried in the response
    const char* exposeHeader_1[2]  = {"hello", "world"};

    memset(&bucketCorsConf, 0, sizeof(obs_bucket_cors_conf));
    bucketCorsConf.id = id_1;
    bucketCorsConf.max_age_seconds = max_age_seconds;
    bucketCorsConf.allowed_method = allowedMethod_1;
    bucketCorsConf.allowed_method_number = 5;
    bucketCorsConf.allowed_origin = allowedOrigin_1;
    bucketCorsConf.allowed_origin_number = 2;
    bucketCorsConf.allowed_header = allowedHeader_1;
    bucketCorsConf.allowed_header_number = 2;
    bucketCorsConf.expose_header = exposeHeader_1;
    bucketCorsConf.expose_header_number = 2;

    set_bucket_cors_configuration(&option, &bucketCorsConf, 1, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("set_bucket_cors success.\n");
    }
    else {
        printf("set_bucket_cors faied(%s).\n", obs_get_status_name(ret_status));
    }
}
```

NOTE

allowed_origin, **allowed_method**, and **allowed_header** can contain at most one wildcard (*). Wildcard characters (*) indicate that all origins, operations, or headers are allowed.

13.3 Viewing CORS Rules

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can call **get_bucket_cors_configuration()** to view CORS rules of a bucket. Sample code is as follows:

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
handler	obs_cors_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample Code

```
static void test_get_cors_config()
{
    obs_options option;
    obs_status  ret_status = OBS_STATUS_BUTT;
    // Set option.
    init_obs_options(&option);
    option->bucket_options.hostName = "<your-endpoint>";
    option->bucket_options.bucketName = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option->bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option->bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // Set callback function.
    obs_cors_handler cors_handler_info =
    {
        {&response_properties_callback, &response_complete_callback},
        &get_cors_info_callback
    };

    get_bucket_cors_configuration(&option, &cors_handler_info, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("get_cors_config success.\n");
    }
    else {
        printf("get_cors_config faied(%s).\n", obs_get_status_name(ret_status));
    }
}
```


13.4 Deleting CORS Rules

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can call **delete_bucket_cors_configuration()** to delete CORS rules of a bucket. Sample code is as follows:

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
handler	obs_response_handler*	Mandatory	Callback function
callback_data	void *	Optional	Callback data

```
static void test_delete_cors_config()
{
    obs_options option;
    obs_status  ret_status = OBS_STATUS_BUTT;
    // Set option.
    init_obs_options(&option);
    option->bucket_options.hostName = "<your-endpoint>";
    option->bucket_options.bucketName = "<Your bucketname>";


    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option->bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option->bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set callback function.
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };

    delete_bucket_cors_configuration(&option, &response_handler, &ret_status);
```



```
    if (OBS_STATUS_OK == ret_status) {
        printf("delete_cors_config success.\n");
    }
    else
    {
        printf("delete_cors_config faied(%s).\n", obs_get_status_name(ret_status));
    }
}
```

14 Setting Access Logging

14.1 Logging Overview

NOTICE

If you have any questions during the development, post them on the **Issues** page of GitHub.

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

For more information, see **Logging**.

14.2 Enabling Bucket Logging

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can call **set_bucket_logging_configuration_obs** to enable bucket logging.

NOTICE

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

NOTE

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

Sample code:

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
target_bucket	char *	Mandatory	When enabling the logging function, the owner of the bucket being logged can specify a target bucket to store the generated log files. Log files generated for multiple buckets can be stored in the same target bucket. If you do so, you need to specify different **target_prefix** to classify logs for different buckets.
target_prefix	char *	Mandatory	You can specify a prefix using this element so that log files are named with this prefix.

Field	Type	Mandatory or Optional	Description
agency	char *	This parameter is mandatory when enabling bucket logging. Do not select this parameter when disabling bucket logging.	Name of the agency created by the owner of the logging bucket for uploading log files by OBS.
acl_group	obs_acl_group *	Optional	Structure of the permission information group
acl_group->acl_grant_count	int	Optional	The number of returned **obs_acl_grant**.
acl_group->acl_grants	obs_acl_grant *	Optional	Pointer to the permission information structure. For details, see **Table 5-2** in **Managing Bucket ACLs**.
handler	obs_response_handler*	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample Code

```
static void test_set_bucket_logging_configuration()
{
    // Grant the WRITE and READ_ACP permissions on the target bucket to the log delivery group.
    set_log_delivery_acl(bucket_name_target);
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
```



```
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // Configure bucket logging.
    set_bucket_logging_configuration_obs(&option, bucket_name_target, "prefix-log",  "Your agency",
        NULL, &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("set bucket(%s) logging successfully. \n",bucket_name_src);
    }
    else
    {
        printf("set bucket logging faied(%s).\n",obs_get_status_name(ret_status));
    }
}
```

Setting ACLs for Objects to Be Logged

Sample code:

```
static void test_set_bucket_logging_configuration_acl()
{
    // Grant the WRITE and READ_ACP permissions on the target bucket to the log delivery group.
    set_log_delivery_acl(bucket_name_target);
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";
    //Read the AK/SK from environment variables.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // Set response callback function.
    obs_response_handler response_handler =
    {
        0,
        &response_complete_callback
    };
    int aclGrantCount = 2;
    obs_acl_grant acl_grants[2] = {0};
    // Grant the FULL_CONTROL permission on logs to the authorized users.
    acl_grants[0].grantee_type = OBS_GRANTEE_TYPE_CANONICAL_USER;
    strcpy(acl_grants[0].grantee.canonical_user.id, "userid1");
    strcpy(acl_grants[0].grantee.canonical_user.display_name, "dis1");
    acl_grants[0].permission = OBS_PERMISSION_READ ;
    // Grant the READ permission on the objects to be logged to all users.
    acl_grants[1].grantee_type = OBS_GRANTEE_TYPE_ALL_OBS_USERS;
    acl_grants[1].permission = OBS_PERMISSION_READ ;
    obs_acl_group g;
    g.acl_grants = acl_grants;
    g.acl_grant_count = aclGrantCount;
    // Configure bucket logging.
    set_bucket_logging_configuration_obs(&option, bucket_name_target, "prefix-log",  "Your agency",
        &g, &response_handler, &ret_status);
```



```
    if (ret_status == OBS_STATUS_OK) {
        printf("set bucket(%s) logging successfully. \n", bucket_name_src);
    }
    else
    {
        printf("set bucket logging faied(%s).\n",obs_get_status_name(ret_status));
    }
}
```

14.3 Viewing Bucket Logging Configuration

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can call **get_bucket_logging_configuration** to view the logging configuration of a bucket. Sample code is as follows:

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
logging_message_data	bucket_logging_message *	Mandatory	Current bucket logging management configuration. For details about **bucket_logging_message**, see the following table.
handler	obs_response_handler*	Mandatory	Callback function
callback_data	void *	Optional	Callback data

The following table describes the bucket logging management configuration structure **bucket_logging_message**.

Field	Type	Description
target_bucket	char *	When enabling the logging function, the owner of the bucket being logged can specify a target bucket to store the generated log files. Log files generated for multiple buckets can be stored in the same target bucket. If you do so, you need to specify different **target_prefix** to classify logs for different buckets.
target_bucket_size	int	Total size of **target_bucket**
target_prefix	char *	You can specify a prefix using this element so that log files are named with this prefix.
target_prefix_size	int	Total size of **target_prefix**
acl_grants	obs_acl_grant *	Pointer to the permission information structure.
acl_grant_count	int *	Pointer to the number of returned **acl_grants**
agency	char *	Name of the agency created by the owner of the logging bucket for uploading log files by OBS
agency_size	int	Total size of **agency**

Sample Code

```
static void test_get_bucket_logging_configuration()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_response_handler response_handler =
    {
        &response_properties_callback, &response_complete_callback
    };
    // Initialize the logging configuration structure.
    bucket_logging_message logging_message;
```



```
    init_bucket_get_logging_message(&logging_message);
    // Obtain bucket logs.
    get_bucket_logging_configuration(&option, &response_handler, &logging_message, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {
        if (logging_message.target_bucket)
        {
            printf("Target_Bucket: %s\n", logging_message.target_bucket);
            if ( logging_message.target_prefix)
            {
                printf("Target_Prefix: %s\n",  logging_message.target_prefix);
            }
            if (logging_message.agency && logging_message.agency[0] != '\0')
            {
                printf(" Agency: %s\n",  logging_message.agency);
            }
            print_grant_info(*logging_message.acl_grant_count, logging_message.acl_grants);
        }
        else
        {
            printf("Service logging is not enabled for this bucket.\n");
        }
    }
    else
    {
        printf("get bucket logging faied(%s).\n", obs_get_status_name(ret_status));
    }
    // Destroy the logging configuration structure.
    destroy_logging_message(&logging_message);
}
```

14.4 Disabling Bucket Logging

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can call **set_bucket_logging_configuration_ob** to delete all logs of a bucket to disable logging of the bucket. Sample code is as follows:

```
static void test_close_bucket_logging_configuration()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
```



```
    // Disable bucket logging.
    set_bucket_logging_configuration_obs(&option, NULL, NULL, NULL,
        NULL, &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("close bucket(%s) logging successfully. \n",bucket_name_src);
    }
    else
    {
        printf("close bucket logging failed(%s).\n",obs_get_status_name(ret_status));
    }
}
```


15 Static Website Hosting

15.1 Static Website Hosting Overview

NOTICE

If you have any questions during the development, post them on the **Issues** page of GitHub.

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

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

15.2 Website File Hosting

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can perform the following to implement website file hosting:

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

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

Step 3 Access the object using a browser.

----End

Sample code:

```
static void test_put_object()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Initialize put_properties which can be used to set object properties.
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    // Set the MIME type.
    put_properties.content_type = "text/html";
    // Set the object ACL to public-read.
    put_properties.canned_acl = OBS_CANNED_ACL_PUBLIC_READ
    // Callback data
    put_file_object_callback_data data;
    memset(&data, 0, sizeof(put_file_object_callback_data));
    // Read the file to be uploaded to the callback data.
    data.infile = 0;
    data.noStatus = 1;
    content_length = read_bytes_from_file("<Uploaded filename>", &data);
    // Callback function
    obs_put_object_handler putobjectHandler =
    {
        { &response_properties_callback, &response_complete_callback },
        &put_buffer_object_data_callback
     };
    // Upload data streams.
    put_object(&option,"<object key>", content_length, &put_properties, 0, &putobjectHandler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("put object from file successfully. \n");
    }
    else
    {
        printf("put object failed(%s).\n",
               obs_get_status_name(data.ret_status));
    }
}
```

15.3 Setting Website Hosting

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can call **set_bucket_website_configuration** to set website hosting on a bucket.

Configuring the Default Homepage, Error Pages and Redirection Rules

The following code shows how to configure the default home page, error pages, and redirection rules. The following table describes the parameters.

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
set_bucket_redirect_all	obs_set_bucket_redirect_all_conf *	Mandatory	Indicates the redirection configuration.
set_bucket_redirect_all->host_name	const char *	Mandatory	Indicates the name of the host where requests will be redirected.
set_bucket_redirect_all->protocol	const char *	Optional	The HTTP or HTTPS protocol used in redirecting requests. The default protocol is HTTP.
set_bucket_website_conf	obs_set_bucket_website_conf *	Mandatory	Describes the configuration of the **website** element in the redirection rules.
set_bucket_website_conf->suffix	const char *	Mandatory	Suffix that is appended to a request initiated for a directory on the website endpoint. For example, if the suffix is **index.html** and you request for **samplebucket/images/**, the data that is returned will be the object with the key name **images/index.html** in the **samplebucket** bucket. Suffix cannot be empty or contain slashes (/).
set_bucket_website_conf->key	const char *	Optional	Indicates the object name that is used when a 4*XX* error occurs. This element identifies the page that is returned when a 4*XX* error occurs.
set_bucket_website_conf->routingrule_info	bucket_website_routingrule *	Optional	For details about redirection rules, see the following table.

Field	Type	Mandatory or Optional	Description
set_bucket_website_conf->routingrule_count	int	Optional	Total size of **set_bucket_website_conf.routingrule_info**
handler	obs_response_handler*	Mandatory	Callback function
callback_data	void *	Optional	Callback data

The following table describes the redirection rule structure **bucket_website_routingrule**.

Field	Type	Mandatory or Optional	Description
key_prefix_equals	const char *	Optional	Indicates the object name prefix when the redirection is applied.
http_errorcode_returned_equals	const char *	Optional	Indicates HTTP error codes when the redirection takes effect. The specified redirection is applied only when the error code returned equals this value.
protocol	const char *	Optional	Indicates protocol used in the redirection request.
host_name	const char *	Optional	Indicates the host name used in the redirection request.
replace_key_prefix_with	const char *	Optional	Indicates the object name prefix used in the redirection request.
replace_key_with	const char *	Optional	Indicates the object name used in the redirection request.

Field	Type	Mandatory or Optional	Description
http_redirect_code	const char *	Optional	Indicates the HTTP status code returned after the redirection request.

```
static void test_set_bucket_website_configuration()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    obs_set_bucket_website_conf set_bucket_website_conf;
    // Configure the default homepage.
    set_bucket_website_conf.suffix = "index.html";
    // Configure the error pages.
    set_bucket_website_conf.key = "Error.html";
    // Define redirection rules.
    set_bucket_website_conf.routingrule_count = 2;
    bucket_website_routingrule temp[2];
    memset(&temp[0], 0, sizeof(bucket_website_routingrule));
    memset(&temp[1], 0, sizeof(bucket_website_routingrule));
    set_bucket_website_conf.routingrule_info = temp;
    temp[0].key_prefix_equals = "key_prefix1";
    temp[0].replace_key_prefix_with = "replace_key_prefix1";
    temp[0].http_errorcode_returned_equals="404";
    temp[0].http_redirect_code = NULL;
    temp[0].host_name = "www.example.com";
    temp[0].protocol = "http";
    temp[1].key_prefix_equals = "key_prefix2";
    temp[1].replace_key_prefix_with = "replace_key_prefix2";
    temp[1].http_errorcode_returned_equals="404";
    temp[1].http_redirect_code = NULL;
    temp[1].host_name = "www.example.com";
    temp[1].protocol = "http";
    // Set response callback function.
    obs_response_handler response_handler =
    {
        0,
        &response_complete_callback
    };
    // Set redirection rules.
    set_bucket_website_configuration(&option, NULL, &set_bucket_website_conf,
        &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("set bucket website conf successfully. \n");
    }
    else
```



```
    {
        printf("set bucket website conf failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

Configuring Redirection for All Requests

Sample code:

```
static void test_set_bucket_website_all()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // Configure redirection for all requests.
    obs_set_bucket_redirect_all_conf set_bucket_redirect_all;
    set_bucket_redirect_all.host_name = "www.example.com";
    set_bucket_redirect_all.protocol = "https";
    set_bucket_website_configuration(&option, &set_bucket_redirect_all, NULL,
        &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("set bucket website all successfully. \n");
    }
    else
    {
        printf("set bucket website all failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

15.4 Viewing Website Hosting Settings

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

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

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
handler	obs_get_bucket_websiteconf_handler *	Mandatory	Callback function
handler->response_handler	obs_response_handler	Mandatory	Callback function
handler->get_bucket_website_conf_callback	obs_get_bucket_websiteconf_callback *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample Code

```
static void test_get_bucket_website_configuration()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_get_bucket_websiteconf_handler response_handler =
    {
        {&response_properties_callback, &response_complete_callback},
          &get_bucket_websiteconf_callback
    };
    // Obtain the hosting settings of a bucket.
    get_bucket_website_configuration(&option, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("get bucket website successfully.\n");
    }
    else
    {
        printf("get bucket website failed(%s).\n", obs_get_status_name(ret_status));
```



```
    }
}
```

15.5 Deleting Hosting Settings

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

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

Parameter Description

Field	Type	Mandatory or Optional	Description
option	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample Code

```
static void test_delete_bucket_website_configuration()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // Set response callback function.
    obs_response_handler response_handler =
    {
```



```
        &response_properties_callback,
        &response_complete_callback
    };
    // Delete the hosting settings of a bucket.
    delete_bucket_website_configuration(&option, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("delete bucket website successfully.\n");
    }
    else
    {
        printf("delete bucket website failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```


16 Tag Management

16.1 Tagging Overview

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

Tags are used to identify and classify OBS buckets.

16.2 Setting Bucket Tags

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can call **set_bucket_tagging** to set the bucket tags.

Parameter Description

Field	Type	Mandatory or Optional	Description
options	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
tagging_list	obs_name_value *	Mandatory	Tag list

Field	Type	Mandatory or Optional	Description
number	unsigned int	Mandatory	Number of tags
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

The following table describes the tag list structure **obs_name_value**.

Field	Type	Description
name	char *	Tag key
value	char *	Tag value

Sample Code

```
static void test_set_bucket_tagging()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // Set response callback function.
    obs_response_handler response_handler =
    {
        0,
        &response_complete_callback
    };
    // Define the bucket tag.
    char tagKey[][OBS_COMMON_LEN_256] = {{"k1"},{"k2"},{"k3"},{"k4"},{"k5"},{"k6"},{"k7"},{"k8"},{"k9"},{"k10"}};
    char tagValue[][OBS_COMMON_LEN_256] = {{"v1"},{"v2"},{"v3"},{"v4"},{"v5"},{"v6"},{"v7"},{"v8"},{"v9"},{"v10"}};
    obs_name_value tagginglist[10] = {0};
    int i=0;
    for(;i<10;i++)
    {
        tagginglist[i].name = tagKey[i];
        tagginglist[i].value  = tagValue[i];
    }
    // Set bucket tags.
    set_bucket_tagging(&option, tagginglist, 8, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("set bucket tagging successfully. \n");
```



```
    }
    else
    {
        printf("set bucket tagging failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

NOTE

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

16.3 Viewing Bucket Tags

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can call **get_bucket_tagging** to view bucket tags.

Parameter Description

Field	Type	Mandatory or Optional	Description
options	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter
handler	obs_get_bucket_tagging_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

The following table describes the callback function types of **obs_get_bucket_tagging_handler**.

Field	Type	Description
response_handler	obs_response_handler	Callback function handler for response
get_bucket_tagging_callback	obs_get_bucket_tagging_callback *	Callback function for obtaining bucket tags

Sample Code

```
static void test_get_bucket_tagging()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";
    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

        // Set response callback function.
    obs_get_bucket_tagging_handler response_handler =
    {
        {&response_properties_callback, &get_bucket_tagging_complete_callback},
          &get_bucket_tagging_callback
    };
    // Create callback data.
    TaggingInfo tagging_info;
    memset(&tagging_info, 0, sizeof(TaggingInfo));
    tagging_info.ret_status = OBS_STATUS_BUTT;
    // Obtain bucket tags
    get_bucket_tagging(&option, &response_handler, &tagging_info);
    if (OBS_STATUS_OK == tagging_info.ret_status) {
        printf("get bucket tagging successfully.\n");
    }
    else
    {
        printf("get bucket tagging failed(%s).\n", obs_get_status_name(tagging_info.ret_status));
    }
}
```

16.4 Deleting Bucket Tags

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

You can call **delete_bucket_tagging** to delete bucket tags.

Parameter Description

Field	Type	Mandatory or Optional	Description
options	The context of the bucket. For details, see **Configuring option**.	Mandatory	Bucket parameter

Field	Type	Mandatory or Optional	Description
handler	obs_response_handler *	Mandatory	Callback function
callback_data	void *	Optional	Callback data

Sample Code

```
static void test_delete_bucket_tagging()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // Set response callback function.
    obs_response_handler response_handler =
    {
        &response_properties_callback,
        &response_complete_callback
    };
    // Delete bucket tags.
    delete_bucket_tagging(&option, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("delete bucket tagging successfully.\n");
    }
    else
    {
        printf("delete bucket tagging failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```


17 Server-Side Encryption

17.1 Server-Side Encryption Overview

NOTICE

If you have any questions during the development, post them on the **Issues** page of GitHub.

OBS supports server-side encryption.

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

17.2 Encryption Description

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

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

API Method in OBS C SDK	Description	Supported Encryption Type	Transmission Protocol
put_object	Sets the encryption algorithm and key during object upload to enable server-side encryption.	SSE-KMS SSE-C	HTTPS

API Method in OBS C SDK	Description	Supported Encryption Type	Transmission Protocol
get_object	Sets the decryption algorithm and key during object download to decrypt the object.	SSE-KMS SSE-C	HTTPS
copy_object	1. Sets the decryption algorithm and key for decrypting the source object during object copy. 2. Sets the encryption algorithm and key during object copy to enable the encryption algorithm for the target object.	SSE-KMS SSE-C	When the target object is a non-encrypted object: HTTP or HTTPS. HTTPS in other cases.
get_object_metadata	Sets the decryption algorithm and key when obtaining the object metadata to decrypt the object.	SSE-KMS SSE-C	When the encryption type is SSE-KMS: HTTP or HTTPS. HTTPS in other cases.
initiate_multi_part_upload	Sets the encryption algorithm and key when initializing a multipart upload to enable server-side encryption for the final object generated.	SSE-KMS SSE-C	HTTPS
upload_part	Sets the encryption algorithm and key during multipart upload to enable server-side encryption for parts.	SSE-KMS SSE-C	HTTPS
complete_multi_part_upload	Sets the encryption algorithm and key during part combination to enable server-side encryption after it.	SSE-KMS SSE-C	HTTP or HTTPS

API Method in OBS C SDK	Description	Supported Encryption Type	Transmission Protocol
copy_part	1. Sets the decryption algorithm and key for decrypting the source object during multipart copy. 2. Sets the encryption algorithm and key during multipart copy to enable the encryption algorithm for the target part.	SSE-KMS SSE-C	HTTPS

Parameter Description

The following table describes the encryption and decryption parameters for **server_side_encryption_params**.

Field	Type	Description
encryption_type	obs_encryption_type	Encryption mode: **OBS_ENCRYPTION_KMS**: SE-KMS; **OBS_ENCRYPTION_SSEC**: SSE-C
kms_server_side_encryption	char *	Indicates that SSE-KMS is used. Objects are encrypted using SSE-KMS.
kms_key_id	char *	Indicates the master key ID of an encrypted object. This parameter is used in SSE-KMS mode. If the customer does not provide the master key ID, the default master key ID will be used.
ssec_customer_algorithm	char *	Indicates the algorithm used to encrypt an object. The parameter is used in SSE-C mode.

Field	Type	Description
ssec_customer_key	char *	Indicates the key used to encrypt an object. The parameter is used in SSE-C mode.
des_ssec_customer_algorithm	char *	Indicates the algorithm used to decrypt a source object. The parameter is used in SSE-C mode.
des_ssec_customer_key	char *	Indicates the key used to decrypt a source object. The parameter is used in SSE-C mode.

17.3 Example of Encryption

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub.

Encrypting an Object to Be Uploaded

Sample code:

```
static void test_put_object_by_aes_encrypt()
{
    // Buffer to be uploaded
    char *buffer = "11111111";
    // Length of the buffer to be uploaded
    int buffer_size = strlen(buffer);
    // Name of an object to be uploaded
    char *key = "put_buffer_aes";
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    option.bucket_options.protocol = OBS_PROTOCOL_HTTPS;
    // Initialize the properties of an object to be uploaded.
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    // Initialize the structure for storing uploaded data.
    put_buffer_object_callback_data data;
```



```
    memset(&data, 0, sizeof(put_buffer_object_callback_data));
    data.put_buffer = buffer;
    data.buffer_size = buffer_size;
    // Server-side encryption
    server_side_encryption_params encryption_params;
    memset(&encryption_params, 0, sizeof(server_side_encryption_params));
    encryption_params.ssec_customer_algorithm = "AES256";
    encryption_params.ssec_customer_key =
            "K7QkYpBkM5+hcs27fsNkUnNVaobncnLht/rCB2o/9Cw=";
    // Set callback function.
    obs_put_object_handler putobjectHandler =
    {
       { &response_properties_callback, &put_buffer_complete_callback },
          &put_buffer_data_callback
    };
    put_object(&option, key, buffer_size, &put_properties,
              &encryption_params,&putobjectHandler,&data);

    if (OBS_STATUS_OK == data.ret_status) {
       printf("put object by_aes_encrypt successfully. \n");
    }
    else
    {
       printf("put object by_aes_encrypt encryption failed(%s).\n",
                obs_get_status_name(data.ret_status));
    }
}
```

Decrypting a To-Be-Download Object

Sample code:

```
static void test_get_object_by_aes_encrypt()
{
    char *file_name = "./test_by_aes";
    char *key = "put_buffer_aes";
    obs_object_info object_info;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 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 ACCESS_KEY_ID and SECRET_ACCESS_KEY.
    // Obtain an AK/SK pair on the management console. For details, see https://support.huaweicloud.com/intl/en-us/usermanual-ca/ca_01_0003.html.
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    option.bucket_options.protocol = OBS_PROTOCOL_HTTPS;

    // The SSE key must be transferred during the download of SSE encrypted object.
    server_side_encryption_params encryption_params;
    memset(&encryption_params, 0, sizeof(server_side_encryption_params));
    encryption_params.use_ssec = '1';
    encryption_params.ssec_customer_algorithm = "AES256";
    encryption_params.ssec_customer_key = "K7QkYpBkM5+hcs27fsNkUnNVaobncnLht/rCB2o/9Cw=";

    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key =key;

    get_object_callback_data data;
    data.ret_status = OBS_STATUS_BUTT;
    data.outfile = write_to_file(file_name);

    obs_get_conditions getcondition;
    memset(&getcondition, 0, sizeof(obs_get_conditions));
```



```
    init_get_properties(&getcondition);

    obs_get_object_handler get_object_handler =
    {
        { NULL, &get_object_complete_callback},
        &get_object_data_callback
    };

    get_object(&option, &object_info, &getcondition, &encryption_params,
               &get_object_handler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("get object by_aes successfully . \n");
    }
    else
    {
        printf("get object by_aes faied(%s).\n", obs_get_status_name(data.ret_status));
    }
    fclose(data.outfile);
}
```

18 Troubleshooting

18.1 OBS Server-Side Error Codes

If the OBS server encounters an error when processing a request, a response containing the error code and error description is returned. For details about error codes, their descriptions, and HTTP status codes, see **OBS API Error Codes**.

18.2 SDK Error Handling

SDK errors include the errors returned during the check of function parameters and those returned by the OBS server.

SDK error handling information:

- obs_status: Error code
- obs_get_status_name(): Obtaining the error description
- obs_status_is_retryable(): Checking whether the error code requires service retry

18.3 Log Analysis

Log Path

The OBS C SDK log path is specified by the **LogPath** field in **OBS.ini**. By default, logs are stored in the **logs** directory at the same level as the **lib** directory of the C SDK dynamic library. To locate a fault, view **eSDK-OBS-API-*-C.run.log** or **obs-sdk-c.run.log** in the **logs** directory.

OBS.ini must be in the same directory as **libeSDKLogAPI.so**.

Log Format

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


```
Run logs
2018-05-15 22:22:54 803| INFO|[140677572568864]|request_perform start
```

Log Levels

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(0)** logs and the least information in **ERROR(3)** logs.

Log level description:

- DEBUG(0): Debug level. If this level is set, logs at the **INFO** level and some debugging information will be printed.
- INFO(1): Information level. If this level is set, logs at the **WARN** level, calling process and key information of OBS APIs will be printed.
- **WARN(2)**: Warning level. If this level is set, logs at the **ERROR** level and some critical events, such as **curl_global_init** initialization fail, will be printed.
- **ERROR(3)**: Error level. If this level is set, only error information will be printed.

Enabling System Logging

In the **lib** directory, modify **OBS.ini**, modify the size, number, and level of logs. (The ***_Run** parameter is the most common configuration item.)

```
;Every line must be less than 1024
[LogConfig]
;Log Size: unit=KB, 10MB = 10KB * 1024 = 10240KB
LogSize_Interface=10240
LogSize_Operation=10240
LogSize_Run=10240
;Log Num
LogNum_Interface=10
LogNum_Operation=10
LogNum_Run=10
;Log level: debug = 0,info = 1,warn = 2,error = 3
LogLevel_Interface=0
LogLevel_Operation=0
LogLevel_Run=0
;LogFilePermission
LogFilePermission=0600
[ProductConfig]
;Product Name
sdkname=eSDK-OBS-API-Linux-C
[LogPath]
;Log Path is relative to the path of configuration file
LogPath=../logs
```

Other Configurations

In Windows, the **LogPath** field in the **OBS.ini** can be read in type wchar_t. To do that, you need to set the encoding for the file path first. An example is given here:

set_file_path_code(UNICODE_CODE);//**ANSI_CODE** is used by default.

In addition, the local file path for functions listed in the table below must also be in type wchar_t (parameters are passed in type char* and processed into type wchar_t internally).

Table 18-1

Related Function	Description
download_file	The **downLoad_file** and **check_point_file** members of the **download_file_config** parameter must be in type wchar_t. Pass them in type char*.
upload_file	The **upload_file** and **check_point_file** members of the **upload_file_config** parameter must be in type wchar_t. Pass them in type char*.
set_obs_log_path	The **log_path** parameter must be in type wchar_t. Pass it in type char*.

19 FAQs

19.1 Invalid Proxy Settings

- When the proxy is configured in a Windows SDK demo, the program reports the following error and the proxy configuration fails.

  ```
  Run-Time Check Failure #2 - Stack around the variable 'ret_status' was corrupted.
  ```

  Root cause: Due to asynchronous updates, the demo header file **eSDKOBS.h** of certain SDK versions diverges from the SDK **eSDKOBS.h**, rendering the proxy settings in the option invalid.

  Solution:

  a. Replace *yourSDKpath***source\eSDK_OBS_API\eSDK_OBS_API_C++\inc\eSDKOBS.h** with *yourSDKpath***source\eSDK_OBS_API\eSDK_OBS_API_C++\build\obs\demo\eSDKOBS.h**.

  b. Modify the demo as instructed here to adapt to the changes of **eSDKOBS.h**. (The adaptation of version 3.22.7 is used as an example. The adaptation process of other versions may differ.)

  In the *yourSDKpath***source\eSDK_OBS_API\eSDK_OBS_API_C++\build\obs\demo\ demo_windows.cpp** file, in line 4749, add **obs_upload_file_server_callback server_callback**; in line 4750, add **, server_callback** after the fourth parameter in the **upload_file** function. See the following figure.

  ```
  };
  obs_upload_file_server_callback server_callback;
  upload_file(&option, key, 0, &uploadFileInfo, server_callback, &Handler, 0);
  if (statusG == OBS_STATUS_OK) {
      printf("test upload file successfully. \n");
  }
  else
  {
  printf("test upload file faied.\n");
  printError();
  return ;
  }
  }
  ```


- The proxy is configured but the connection still fails.

  Root cause: The proxy might not be configured in the **get_api_version** function of the SDK **request.c**.

  Solution:

  Configure the proxy (add the **CURLOPT_PROXY** and **CURLOPT_PROXYUSERPWD** items for curl) in the **get_api_version** function by referring to the method for configuring the proxy in the **setup_curl** function of the SDK **request.c**.

