Copyright © Huawei Technologies Co., Ltd. 2019. All rights reserved.
No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.

Trademarks and Permissions

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

Notice

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

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

1.1 SDK Download

- Latest OBS C SDK version: [OBS C SDK]

1.2 Compatibility

- 3.*.* is compatible with 3.0.0.
- 3.*.* is incompatible with 2.*.*.
- 3.*.* is incompatible with 1.*.*
2 Quick Start

2.1 Before You Start

- Ensure that you are familiar with OBS basic concepts, such as bucket, object, and 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 is successful. If yes, the operation fails and you should obtain the fault information from 17.2 SDK Error Handling.

2.2 Setting Up an OBS Environment

**Step 1** Register a cloud service account.

Ensure that you have a cloud service account before using OBS.

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

- An AK is an Access Key ID on OBS. One AK maps to only one user but one user can have multiple AKs. The OBS system recognizes the users who access the system by their access key IDs.
- An SK is the Secret Access Key on OBS, which is required as the key to access OBS. You can generate authentication information based on SKs and request header fields. An SK corresponds to an AK.

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

### Permanent access keys:

1. Log in to OBS Console.
2. In the upper right corner of the page, click the username and choose My Credentials.
3. On the My Credentials page, click the Access Keys tab, and then click Add Access Key below the displayed access key list.
4. In the Add Access Key dialog box that is displayed, enter the password and its verification code.
   
   **NOTE**
   
   - If you have not bound an email address or mobile number, you need to enter only the 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.
5. Click OK.
6. In the Download Access Key dialog box that is displayed, click OK to save the access keys to your browser's default download path.
7. Open the downloaded credentials.csv file to obtain the AK/SK pair.
   
   **NOTE**
   
   - Each user can create up to two valid AK/SK pairs.
   - To prevent the AK/SK from being leaked, keep them secure. If you click Cancel in the dialog box, the access keys will not be downloaded, and you cannot download them later. Create another pair if you need to use them later.

### Temporary access keys:

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

For details about how to obtain temporary access keys, see Obtaining a Temporary AK/SK.
For details about how to use temporary access keys, see Sample code for creating and initializing option using temporary access keys.

---End

2.3 Service Address

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

2.4 Initializing option

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):

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // Set 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);
}
```

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

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    option.bucket_options.token = "<Your SecurityToken>";
    // Set 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);
}
```

2.5 Creating a Bucket

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

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // Set 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 4.1 Creating a Bucket.

NOTICE

- This parameter is not required if the endpoint belongs to the default region (cn-north-1). Otherwise, set this parameter to the region to which the endpoint belongs. Valid regions include: cn-north-1, cn-east-2, cn-south-1, and ap-southeast-1.
- When creating a bucket, you can specify its region. For details, see Creating a Bucket with Parameters Specified.

2.6 Uploading an Object

The data flow is saved to callback_data (see the callback_data definition in 5.1 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 which is parameter character pointer of the callback function. Sample code:

```c
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.hostName = "<your-endpoint";
    option.bucket_options.bucketName = "<Your bucket name>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
```
// 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 5.1 Performing a Streaming Upload.

2.7 Downloading an Object

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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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,
2.8 Listing Object

Sample code:

```c
static void test_list_bucket_objects()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";

    // 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**

For more information, see 6.1 Downloading an Object.

2.9 Deleting an Object

Sample code:

```c
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.hostName = "<your-endpoint>";
option.bucket_options.bucketName = "<Your bucketname>";
option.bucket_options.accessKeyId = "<Your AK>";
option.bucket_options.secretAccessKey = "<Your SK>";
// Set response callback function.
obs_response_handler resqonseHandler =
{
    &response_properties_callback,
    &response_complete_callback
};
// Delete an object.
delete_object(&option,&object_info,&responseHandler, &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 7.4 Deleting Objects.
3 Initialization

3.1 Configuring the AK and SK

To use OBS, you need a valid pair of AK and SK for signature authentication. For details, see 2.2 Setting Up an OBS Environment.

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

3.2 SDK Initialization

ObsClient functions as the C client for accessing OBS. It offers callers a series of APIs for interaction with OBS. These APIs are used for managing and operating resources, such as buckets and objects, stored in OBS. 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 need to call the initialization API obs_initialize. You can only call it once in the main process.

```c
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;
}
```

3.3 Configuring option

When the function of C SDK is called, obs_option must be input. You can use function init_obs_options to initialize the obs_option configuration, and set AK, SK, endpoint, bucket, timeout period, and temporary authentication by using obs_option.obs_options consists of obs_bucket_context and obs_http_request_option. The following table describes the parameters that can be set.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>Recommended Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>host_name</td>
<td>Name of the host where the requested resource is stored. The value can be a domain name of a server.</td>
<td>NULL</td>
<td>-</td>
</tr>
<tr>
<td>bucket_name</td>
<td>Name of the bucket where the operation is performed.</td>
<td>NULL</td>
<td>-</td>
</tr>
<tr>
<td>protocol</td>
<td>The protocol used for sending requests. Possible values include HTTP and HTTPS.</td>
<td>HTTP: OBS_PROTOCOL_HTTP</td>
<td>Default Value</td>
</tr>
<tr>
<td>access_key</td>
<td>AK of OBS.</td>
<td>NULL</td>
<td>-</td>
</tr>
<tr>
<td>secret_access_key</td>
<td>SK used for authentication. It can be used to sign a character string.</td>
<td>NULL</td>
<td>-</td>
</tr>
<tr>
<td>certificate_info</td>
<td>Certificate content used for bidirectional authentication. If this parameter is set to NULL, certificate verification is not performed. If this parameter is not empty, certificate verification is performed.</td>
<td>NULL</td>
<td>-</td>
</tr>
<tr>
<td>obs_storage_class</td>
<td>Set this parameter when the storage type needs to be configured in the PUT or POST request.</td>
<td>OBS Standard: OBS_STORAGE_CLASS_STANDARD</td>
<td>Default Value</td>
</tr>
<tr>
<td>token</td>
<td>Temporary access key SecurityToken</td>
<td>NULL</td>
<td>-</td>
</tr>
</tbody>
</table>
Table 3-2 obs_options.obs_http_request_option parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>Recommended Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>connect_time</td>
<td>Timeout period for establishing an HTTP/HTTPS connection, in ms. The default value is 60,000.</td>
<td>60</td>
<td>[10, 60]</td>
</tr>
<tr>
<td>max_connected_time</td>
<td>Timeout period (in seconds) of an HTTP/HTTPS request. The value 0 indicates that the link is never disconnected.</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>proxy_auth</td>
<td>Proxy authentication information, in the format of username:password.</td>
<td>NULL</td>
<td>-</td>
</tr>
<tr>
<td>proxy_host</td>
<td>Proxy server</td>
<td>NULL</td>
<td>-</td>
</tr>
</tbody>
</table>

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

### 3.4 Configuring SDK Logging

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

### 3.5 Configuring Server-Side Certificate Verification

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

**NOTE**
If the root certificate on the OBS server is issued by an authoritative CA, skip steps 1 and 2. (Root certificates issued by authoritative CAs are in the certificate library of C SDK.)

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

**Step 2**  Save the obs.cer certificate to a specified directory.
Step 3  Applications use HTTPS to communicate with each other. For details about how to call the API, see **Usage of HTTPS Service APIs**.

---End

Usage of HTTPS Service APIs

1. You can call `init_certificate_by_path` before `init_obs_options` to implement HTTPS communication. The root certificate path is specified by a parameter or uses the default system certificate.

Parameter Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>protocol</td>
<td>obs_protocol</td>
<td>Mandatory</td>
<td>Protocol parameter. The value is <code>OBS_PROTOCOL_HTTPS</code> or <code>OBS_PROTOCOL_HTTP</code>.</td>
</tr>
<tr>
<td>ca_conf</td>
<td>obs_certificate_conf</td>
<td>Optional</td>
<td>Certificate configuration policy. The value can be <code>OBS_NO_CERTIFICATE</code>, <code>OBS_DEFAULT_CERTIFICATE</code>, or <code>OBS_DEFINED_CERTIFICATE</code>.</td>
</tr>
<tr>
<td>path</td>
<td>char *</td>
<td>Optional</td>
<td>Certificate path</td>
</tr>
<tr>
<td>path_length</td>
<td>int</td>
<td>Optional</td>
<td>Certificate path length</td>
</tr>
</tbody>
</table>

Sample Code

1. HTTP communication

```c
ret_status = init_certificate_by_path(OBS_PROTOCOL_HTTP, OBS_NO_CERTIFICATE, NULL, 0);
init_obs_options(&option);
```

2. HTTPS communication. Use the default certificate which is `client.pem` in the `lib` directory.

```c
ret_status = init_certificate_by_path(OBS_PROTOCOL_HTTPS, OBS_DEFAULT_CERTIFICATE, NULL, 0);
init_obs_options(&option);
```

3. HTTPS communication. The application specifies the certificate path and length.

```c
ret_status = init_certificate_by_path(OBS_PROTOCOL_HTTPS,
```
2. You can also call \texttt{init_certificate_by_buffer} before \texttt{init_obs_options} to implement HTTPS communication.

### Parameter Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer</td>
<td>char *</td>
<td>Mandatory</td>
<td>Certificate content</td>
</tr>
<tr>
<td>buffer_length</td>
<td>int</td>
<td>Mandatory</td>
<td>The length of the certificate content</td>
</tr>
</tbody>
</table>

### Sample Code

```c
ret_status = init_certificate_by_buffer(ca_buffer, length);
init_obs_options(&option);
```
### 4.1 Creating a Bucket

You can call `create_bucket` to create a bucket.

**Parameter Description**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>option</code></td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td><code>canned_acl</code></td>
<td>obs_canned_acl. See 4.5 Managing Bucket ACLs.</td>
<td>Mandatory</td>
<td>Access control policy</td>
</tr>
<tr>
<td><code>location_constraint</code></td>
<td>char *</td>
<td>Optional</td>
<td>Bucket location</td>
</tr>
<tr>
<td><code>handler</code></td>
<td>obs_response_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td><code>callback_data</code></td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

**Sample Code**

Sample code:

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

option.bucket_options.hostName = "<your-endpoint>";
option.bucket_options.bucketName = "<Your bucketname>";
option.bucket_options.accessKeyId = "<Your AK>";
option.bucket_options.secretAccessKey = "<Your SK>";

// 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_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**

- This parameter is not required if the endpoint belongs to the default region (cn-north-1). Otherwise, set this parameter to the region to which the endpoint belongs. Valid regions include: cn-north-1, cn-east-2, cn-south-1, and ap-southeast-1.
- When creating a bucket, you can specify its region. For details, see Creating a Bucket with Parameters Specified.

---

### Creating a Bucket with Parameters Specified

When creating a bucket, you can specify the ACL, storage class, and location for the bucket. OBS provides three storage classes for buckets. For details, see 4.8 Storage Class. Sample codes are as follows:

```c
obs_status ret_status = OBS_STATUS_BUTT;
// Create and initialize option.
```
```
obs_options option;
init_obs_options(&option);
option.bucket_options.hostName = "<your-endpoint>";
option.bucket_options.bucketName = "<Your bucketname>";
option.bucket_options.accessKeyId = "<Your AK>";
option.bucket_options.secretAccessKey = "<Your SK>";

    // 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));
    }
```

### 4.2 Listing Buckets

You can call `list_bucket_obs` to list buckets.

**Parameter Description**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>handler</td>
<td>obs_list_service_obs_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

**Sample Code**

```
static void test_list_bucket_obs()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    option.bucket_options.certificate_info = "<Your certificate>";
```
list_service_data data;
memset(&data, sizeof(list_service_data), 0, sizeof(list_service_data));

// Self-defined response callback function
obs_list_service_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).
", obs_get_status_name(data.ret_status));
}

\[\text{NOTE}\]

Obtained bucket names are listed in the lexicographical order.

### 4.3 Deleting a Bucket

You can call delete_bucket to delete a bucket.

#### Parameter Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>handler</td>
<td>obs_response_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

#### Sample Code

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

    option.bucket_options.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
```
option.bucket_options.secretAccessKey = "<Your SK>";
// 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 part 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.

### 4.4 Checking Whether a Bucket Exists

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

**Parameter Description**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td></td>
<td>the bucket, see <a href="#">3.3 Configuring Option</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>handler</td>
<td>obs_response_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

**Sample Code**

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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));
}

4.5 Managing Bucket ACLs

A bucket ACL can be configured in three modes:

1. Specify a pre-defined access control policy during bucket creation.
2. Call `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.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Description</th>
<th>Value in OBS C SDK</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>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.</td>
<td>obs_permission. OBS_PERMISSION_READ</td>
</tr>
<tr>
<td>WRITE</td>
<td>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.</td>
<td>obs_permission. OBS_PERMISSION_WRITE</td>
</tr>
<tr>
<td>READ_ACP</td>
<td>A grantee with this permission can obtain the ACL of a bucket or object. A bucket or object owner has this permission permanently.</td>
<td>obs_permission. OBS_PERMISSION_READ_ACP</td>
</tr>
<tr>
<td>Permission</td>
<td>Description</td>
<td>Value in OBS C SDK</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>WRITE_ACP</td>
<td>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.</td>
<td>obs_permission. OBS_PERMISSION_WRITE_ACP</td>
</tr>
<tr>
<td>FULL_CONTROL</td>
<td>A grantee with this permission for a bucket has READ, WRITE, READ_ACP, and WRITE_ACP permissions for the bucket. A grantee with this permission for an object has READ, WRITE, READ_ACP, and WRITE_ACP permissions for the object.</td>
<td>obs_permission. OBS_PERMISSION_FULL_CONTROL</td>
</tr>
</tbody>
</table>

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

**Table 4-1 Pre-defined Access Control Policies**

<table>
<thead>
<tr>
<th>Permission</th>
<th>Description</th>
<th>Value in OBS C SDK</th>
</tr>
</thead>
<tbody>
<tr>
<td>private</td>
<td>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.</td>
<td>obs_canned_acl. OBS_CANNED_ACL_PRIVATE</td>
</tr>
<tr>
<td>Permission</td>
<td>Description</td>
<td>Value in OBS C SDK</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------------------------------------</td>
</tr>
<tr>
<td>public-read</td>
<td>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.</td>
<td>obs_canned_acl. OBS_CANNED_ACL_PUBLIC_READ</td>
</tr>
<tr>
<td>public-read-write</td>
<td>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.</td>
<td>obs_canned_acl. OBS_CANNED_ACL_PUBLIC_READ_WRITE</td>
</tr>
<tr>
<td>public-read-delivered</td>
<td>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.</td>
<td>obs_canned_acl. OBS_CANNED_ACL_PUBLIC_READ_DELIVERED</td>
</tr>
</tbody>
</table>
### Specifying a Pre-defined Access Control Policy During Bucket Creation

Sample code:

```c
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.hostname = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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 | Request for the context of the bucket, see 3.3 Configuring option | Mandatory | Bucket parameter

canned_acl | obs_canned_acl | Mandatory | For details about how to manage bucket access rights, see Table 4-1 in 4.5 Managing Bucket ACLs.

**handler** | obs_response_handler | Mandatory | Callback function

callback_data | void * | Optional | Callback data

**Sample code:**

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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:
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>aclinfo</td>
<td>manager_acl_info *</td>
<td>Mandatory</td>
<td>Structure for managing ACL permission information</td>
</tr>
<tr>
<td>aclinfo-&gt;object_info</td>
<td>obs_object_info *</td>
<td>Ignoring malicious programs</td>
<td>Object name and version number. For non-multi-version objects, set version to 0.</td>
</tr>
<tr>
<td>aclinfo-&gt;owner_id</td>
<td>char *</td>
<td>Ignoring malicious programs</td>
<td>User account ID</td>
</tr>
<tr>
<td>aclinfo-&gt;acl_grant_count_return</td>
<td>int *</td>
<td>Mandatory</td>
<td>Pointer to the number of returned aclinfo-&gt;acl_grants</td>
</tr>
<tr>
<td>aclinfo-&gt;acl_grants</td>
<td>obs_acl_grant *</td>
<td>Mandatory</td>
<td>Pointer to the permission information structure. For details, see Table 4-2.</td>
</tr>
<tr>
<td>aclinfo-&gt;object_delivered</td>
<td>obs_object_delivered</td>
<td>Optional</td>
<td>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_FALSE.</td>
</tr>
<tr>
<td>handler</td>
<td>obs_response_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>
Table 4-2 Description of the permission information structure obs_acl_grant

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantee_type</td>
<td>obs_grantee_type</td>
<td>For details, see Table 4-3.</td>
</tr>
<tr>
<td>grantee.canonical_user_id</td>
<td>char</td>
<td>CanonicalUser ID.</td>
</tr>
<tr>
<td>permission</td>
<td>obs_permission</td>
<td>For details, see Managing Bucket ACLs.</td>
</tr>
<tr>
<td>bucket_delivered</td>
<td>obs_bucket_delivered</td>
<td>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.</td>
</tr>
</tbody>
</table>

Table 4-3 Description of the authorized user type

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>obs_grantee_type.OBS_GRANT EE_TYPE_CANONICAL_USER</td>
<td>The OBS user, bucket, or object permission can be granted to any user who has an OBS account. OBS users can use the AK and SK to access OBS.</td>
</tr>
<tr>
<td>obs_grantee_type.OBS_GRANT EE_TYPE_ALL_USERS</td>
<td>All users can access buckets or objects, including anonymous users.</td>
</tr>
</tbody>
</table>

Sample code:

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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");
    }
} 
```
NOTE
The owner or grantee ID needed in the ACL indicates the account ID, which can be viewed on the My Credential page of OBS Console.

Obtaining the Bucket ACL

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

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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);
}
```

4.6 Obtaining Bucket Storage Information

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>capacity_length</td>
<td>int</td>
<td>Mandatory</td>
<td>Cache size of the used capacity</td>
</tr>
<tr>
<td>capacity</td>
<td>char *</td>
<td>Mandatory</td>
<td>Used capacity</td>
</tr>
<tr>
<td>object_number_length</td>
<td>int</td>
<td>Mandatory</td>
<td>Cache size of the object number</td>
</tr>
<tr>
<td>object_number</td>
<td>char *</td>
<td>Mandatory</td>
<td>Cache of the object number</td>
</tr>
<tr>
<td>handler</td>
<td>obs_response_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

### Sample Code

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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
```
4.7 Bucket Quota

Setting a Bucket Quota

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>storage_quota</td>
<td>uint64_t</td>
<td>Mandatory</td>
<td>Quota size, in bytes.</td>
</tr>
<tr>
<td>handler</td>
<td>obs_response_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

Sample code:

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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\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_quota` function to obtain bucket quotas. The following table describes the parameters.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>storagequota_return</td>
<td>uint64_t *</td>
<td>Mandatory</td>
<td>The obtained quota size, in bytes.</td>
</tr>
<tr>
<td>handler</td>
<td>obs_response_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

Sample code:

```c
static void test_get_bucket_quota(char *bucket_name)
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    init_obs_options(&option);
    option.bucket_options.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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 {
```

Object Storage Service
C SDK Developer Guide
4 Bucket Management

Issue 03 (2019-03-30)  Copyright © Huawei Technologies Co., Ltd.  29
4.8 Storage Class

OBS allows you to set storage classes for buckets. The storage class of an object defaults to 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.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Value in OBS C SDK</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBS Standard</td>
<td>Features low access latency and high throughput and is applicable to storing frequently-accessed (multiple times per month) hotspot or small objects (&lt; 1 MB) requiring quick response.</td>
<td>OBS_STORAGE_CLASS_STANDARD</td>
</tr>
<tr>
<td>OBS Infrequent Access</td>
<td>Is applicable to storing semi-frequently accessed (less than 12 times a year) data requiring quick response.</td>
<td>OBS_STORAGE_CLASS_STANDARD_IA</td>
</tr>
<tr>
<td>OBS Archive</td>
<td>Is applicable to archiving rarely-accessed (once a year) data.</td>
<td>OBS_STORAGE_CLASS_GLACIER</td>
</tr>
</tbody>
</table>

For more information, see `Storage Classes Overview`.

**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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>storage_class_policy</td>
<td>obs_storage_class</td>
<td>Mandatory</td>
<td>Storage class</td>
</tr>
<tr>
<td>handler</td>
<td>obs_response_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>
Sample code:
```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see <a href="#">3.3 Configuring option</a></td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>handler</td>
<td>obs_get_bucket_storage_class_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

Sample code:
```c
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.hostName = "<your-endpoint>";
```
option.bucket_options.bucketName = "<Your bucketname>";
option.bucket_options.accessKeyId = "<Your AK>";
option.bucket_options.secretAccessKey = "<Your SK>";
// 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));
    }
}
5 Uploading an Object

5.1 Performing a Streaming Upload

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>key</td>
<td>char *</td>
<td>Mandatory</td>
<td>Object name</td>
</tr>
<tr>
<td>content_length</td>
<td>uint64_t</td>
<td>Mandatory</td>
<td>Length of the object content</td>
</tr>
<tr>
<td>put_properties</td>
<td>obs_put_properties*</td>
<td>Mandatory</td>
<td>Properties of the uploaded object</td>
</tr>
<tr>
<td>encryption_params</td>
<td>server_side_encryptio n_params *</td>
<td>Optional</td>
<td>Encryption setting of the uploaded object</td>
</tr>
<tr>
<td>handler</td>
<td>obs_put_object_handl er *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>
Sample Code

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    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(\%).\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.

### 5.2 Performing a File-Based Upload

File-based upload uses local files as the data source of objects. For parameters, see 5.1 Performing a Streaming Upload. Sample code:

```c
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.hostName = "<your-endpoint>";
option.bucket_options.bucketName = "<Your bucketname>";
option.bucket_options.secretAccessKey = "<Your SK>";
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.

5.3 Setting Object Properties

You can set properties for an object when uploading it. Object properties include the object length, MIME type, MD5 value (for verification), storage class, and customized metadata. Object attributes can be uploaded in several modes (streaming upload, file-based upload, multipart upload, browser-based upload) or be configured when 7.5 Copying an Object. Parameters to set object properties are in the obs_put_properties structure.

The following table describes object properties.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object length</td>
<td>Indicates the object length. If the object length exceeds the stream or file length, the object will be truncated.</td>
<td>Actual length of the stream or file</td>
</tr>
</tbody>
</table>
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:

```c
static void test_put_object_from_file2()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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);
```
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // Set the storage class to OBS 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;
    // 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));
    }
}
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 4.8 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:

```c
static void test_put_object_from_file4()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.hostname = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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_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));
    }
}
```

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 7.1 Obtaining Object Properties.
- When you use `get_object` to download an object, its customized metadata will also be downloaded.
5.4 Performing a Multipart Upload

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

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

Multipart upload consists of three phases:

**Step 1** Initialize a multipart upload (**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

Initialize a Multipart Upload

Before upload, you need to notify OBS of initializing a multipart upload. 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.

You can use **initiate_multi_part_upload** to initialize a multipart upload task. The following table describes the parameters.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>key</td>
<td>char *</td>
<td>Mandatory</td>
<td>Object name</td>
</tr>
<tr>
<td>upload_id_return_size</td>
<td>int</td>
<td>Mandatory</td>
<td>Size of the cache for storing multipart upload IDs</td>
</tr>
<tr>
<td>upload_id_return</td>
<td>char *</td>
<td>Mandatory</td>
<td>Cache of the multipart upload ID</td>
</tr>
<tr>
<td>put_properties</td>
<td>obs_put_properties*</td>
<td>Optional</td>
<td>Properties of the uploaded object</td>
</tr>
<tr>
<td>encryption_params</td>
<td>server_side_encryptio_n_params *</td>
<td>Optional</td>
<td>Configuring server-side encryption</td>
</tr>
</tbody>
</table>
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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
    };
    // Initialize a multipart upload.
    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 failed(%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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see <a href="#">3.3 Configuring option</a></td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>key</td>
<td>char *</td>
<td>Mandatory</td>
<td>Object name</td>
</tr>
<tr>
<td>upload_part_info</td>
<td>obs_upload_part_info *</td>
<td>Mandatory</td>
<td>Information about the uploading part</td>
</tr>
<tr>
<td>upload_part_info-&gt;part_number</td>
<td>unsigned int</td>
<td>Mandatory</td>
<td>ID of the uploading part The value is an integer from 1 to 10000.</td>
</tr>
<tr>
<td>upload_part_info-&gt;upload_id</td>
<td>char *</td>
<td>Mandatory</td>
<td>Task ID of a multipart upload</td>
</tr>
<tr>
<td>content_length</td>
<td>uint64_t</td>
<td>Mandatory</td>
<td>Length of the uploaded content</td>
</tr>
<tr>
<td>put_properties</td>
<td>obs_put_properties*</td>
<td>Optional</td>
<td>Properties of the uploaded object</td>
</tr>
<tr>
<td>encryption_params</td>
<td>server_side_encryptio n_params *</td>
<td>Optional</td>
<td>Configuring server-side encryption</td>
</tr>
<tr>
<td>handler</td>
<td>obs_upload_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

Sample code:

```c
static void test_upload_part()
{
    // Create and initialize option.
    obs_options *option;
    init_obs_options(&option);
    option.bucket_options.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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;
```
```c
init_put_properties(&put_properties);
    // Callback function
    obs_upload_handler Handler =
    {
        {&response_properties_callback, &response_complete_callback},
        &testup_upload_file_data_callback
    };
    // Initialize callback data.
    test_upload_file_callback_data data =
    {&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;
    data.part_num = (filesize % uploadSize == 0) ? (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, &putProperties, 0, &Handler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("test upload part 1 successfully. \n");
    } else {
        printf("test upload part 1 failed(%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, &putProperties, 0, &Handler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("test upload part 2 successfully. \n");
    } else {
        printf("test upload part 2 failed(%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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see <a href="#">3.3 Configuring option</a></td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>key</td>
<td>char *</td>
<td>Mandatory</td>
<td>Object name</td>
</tr>
<tr>
<td>upload_id</td>
<td>char *</td>
<td>Mandatory</td>
<td>Indicates a multipart upload.</td>
</tr>
<tr>
<td>part_number</td>
<td>unsigned int</td>
<td>Mandatory</td>
<td>Number of segments, <code>complete_upload_Info</code> array length</td>
</tr>
<tr>
<td>complete_upload_Info</td>
<td>obs_complete_upload_Info *</td>
<td>Mandatory</td>
<td>Parts information array</td>
</tr>
<tr>
<td>complete_upload_Info-&gt;part_number</td>
<td>unsigned int</td>
<td>Mandatory</td>
<td>Part number</td>
</tr>
<tr>
<td>complete_upload_Info-&gt;etag</td>
<td>char *</td>
<td>Mandatory</td>
<td>ETag value of a part</td>
</tr>
<tr>
<td>put_properties</td>
<td>obs_put_properties*</td>
<td>Optional</td>
<td>Properties of the uploaded object</td>
</tr>
<tr>
<td>handler</td>
<td>obs_complete_multi_part_upload_handler*</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

Sample code:

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

    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
```
option.bucket_options.hostName = "<your-endpoint>";
option.bucket_options.bucketName = "<Your bucketname>";
option.bucket_options.accessKeyId = "<Your AK>";
option.bucket_options.secretAccessKey = "<Your SK>";

// 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 = "65fe0e61b35c8deead213871033f7fa";
info[1].part_number = "2";
info[1].etag = "0433d5ffca28450ce8b6cf25ab8955267";

// 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:

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    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;
iinitiate_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).
", obs_get_status_name(ret_status));
}
// Concurrent upload
test_concurrent_upload_file_callback_data *concurrent_upload_file =
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);
5.5 Performing a Multipart Copy

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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>key</td>
<td>char *</td>
<td>Mandatory</td>
<td>Object name</td>
</tr>
<tr>
<td>object_info</td>
<td>obs_copy_destination_object_info *</td>
<td>Mandatory</td>
<td>Indicates a multipart upload.</td>
</tr>
<tr>
<td>object_info-destination_bucket</td>
<td>char *</td>
<td>Mandatory</td>
<td>Bucket where the target object is located</td>
</tr>
<tr>
<td>object_info-destination_key</td>
<td>char *</td>
<td>Mandatory</td>
<td>Target object name</td>
</tr>
<tr>
<td>object_info-last_modified_return</td>
<td>int64_t *</td>
<td>Mandatory</td>
<td>Latest time when the object was modified</td>
</tr>
<tr>
<td>object_info-etag_return_size</td>
<td>int</td>
<td>Mandatory</td>
<td>Cache size of ETag</td>
</tr>
<tr>
<td>object_info-etag_return</td>
<td>char *</td>
<td>Mandatory</td>
<td>Cache of ETag</td>
</tr>
<tr>
<td>copypart</td>
<td>obs_upload_part_info *</td>
<td>Mandatory</td>
<td>Information about the part to be uploaded</td>
</tr>
<tr>
<td>copypart-part_number</td>
<td>unsigned int</td>
<td>Mandatory</td>
<td>ID of the part to be uploaded</td>
</tr>
<tr>
<td>copypart-upload_id</td>
<td>char *</td>
<td>Mandatory</td>
<td>Task ID of a multipart upload</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Mandatory or Optional</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------</td>
<td>-----------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>put_properties</td>
<td>obs_put_properties*</td>
<td>Optional</td>
<td>Properties of the uploaded object</td>
</tr>
<tr>
<td>encryption_params</td>
<td>server_side_encryptio n_params *</td>
<td>Optional</td>
<td>Configuring server-side encryption</td>
</tr>
<tr>
<td>handler</td>
<td>obs_response_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

Sample code:

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";

    // 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
    {
```

```c
```
5.6 Performing a Resumable Upload

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

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

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Field</strong></td>
</tr>
<tr>
<td>option</td>
</tr>
<tr>
<td>key</td>
</tr>
<tr>
<td>upload_file_config</td>
</tr>
<tr>
<td>encryption_params</td>
</tr>
</tbody>
</table>
### Field Type Mandatory or Optional Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handler</td>
<td>obs_upload_file_response_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

The following table describes the structure of `obs_upload_file_configuration`.

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>upload_file</td>
<td>char *</td>
<td>Mandatory</td>
<td>Local file to be uploaded</td>
</tr>
<tr>
<td>part_size</td>
<td>uint64_t</td>
<td>Mandatory</td>
<td>Part size, in bytes. The value ranges from 5 MB (default) to 5 GB.</td>
</tr>
<tr>
<td>check_point_file</td>
<td>char *</td>
<td>Mandatory</td>
<td>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.</td>
</tr>
<tr>
<td>enable_check_point</td>
<td>int</td>
<td>Mandatory</td>
<td>Whether to enable the resumable upload mode. The default value is 0, which indicates that this mode is disabled.</td>
</tr>
<tr>
<td>task_num</td>
<td>int</td>
<td>Mandatory</td>
<td>Maximum number of parts that can be concurrently uploaded. The default value is 1.</td>
</tr>
</tbody>
</table>

### Sample Code

Sample code:

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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>";

// Callback function
obs_upload_file_response_handler Handler =
{
   {&response_properties_callback, &response_complete_callback},
   &uploadFileResultCallback
};
upload_file(&option, "<Your Key>", 0, &uploadFileInfo, &Handler, &ret_status);
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));
}

**NOTE**
- The API for resumable upload, which is implemented based on 5.4 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.

5.7 Performing an Appendable Upload

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**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>key</td>
<td>char *</td>
<td>Mandatory</td>
<td>Object name</td>
</tr>
</tbody>
</table>
### Sample Code

```c
static void test_append_object_from_buffer()
{
    // Bufffer 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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    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 buffer size.
    data.buffer_size = buffer_size;
    // Set callback function.
    obs_put_object_handler putobjectHandler =
    {
        &response_properties_callback, &put_buffer_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 (status code 409) if a normal object with the same name exists.
- 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 10000 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 Downloading an Object

6.1 Downloading an Object

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

Parameter Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>object_info</td>
<td>obs_object_info *</td>
<td>Mandatory</td>
<td>Object name and version number. For non-multi-version objects, set <code>version</code> to 0.</td>
</tr>
<tr>
<td>get_conditions</td>
<td>obs_get_conditions *</td>
<td>Mandatory</td>
<td>Sets filter conditions of the object and read range.</td>
</tr>
<tr>
<td>encryption_params</td>
<td>server_side_encryptio n_params *</td>
<td>Optional</td>
<td>Obtains the decryption settings of an object.</td>
</tr>
<tr>
<td>handler</td>
<td>obs_get_object_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

Sample Code

```c
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.hostName = "<your-endpoint>";
option.bucket_options.bucketName = "<Your bucketname>";
option.bucket_options.accessKeyId = "<Your AK>";
option.bucket_options.secretAccessKey = "<Your SK>";

// 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.

6.2 Performing a Conditioned Download

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

You can set the following conditions:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>obs_get_conditions.if_modified_since</code></td>
<td>Returns the source object if it is modified after the time specified by this parameter; otherwise, an exception is thrown.</td>
</tr>
<tr>
<td><code>obs_get_conditions.if_not_modified_since</code></td>
<td>Returns the source object if it remains unchanged since the time specified by this parameter; otherwise, an exception is thrown.</td>
</tr>
<tr>
<td><code>obs_get_conditions.if_match_etag</code></td>
<td>Returns the source object if its ETag is the same as the one specified by this parameter; otherwise, an exception is thrown.</td>
</tr>
<tr>
<td><code>obs_get_conditions.if_not_match_etag</code></td>
<td>Returns the source object if its ETag is different from the one specified by this parameter; otherwise, an exception is thrown.</td>
</tr>
</tbody>
</table>

**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:

```c
static void test_get_object_by_range()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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
    }
```
6.3 Downloading an Archive Object

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Value in OBS C SDK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expedited</td>
<td>Data can be restored within 1 to 5 minutes.</td>
<td>OBS_TIER_EXPEDITED</td>
</tr>
<tr>
<td>Standard</td>
<td>Data can be restored within 3 to 5 hours. This is the default option.</td>
<td>OBS_TIER_STANDARD</td>
</tr>
<tr>
<td>Bulk</td>
<td>Data can be restored within 5 to 12 hours.</td>
<td>OBS_TIER_BULK</td>
</tr>
</tbody>
</table>

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

### Parameter Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see <a href="#">3.3 Configuring option</a></td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>object_info</td>
<td>obs_object_info *</td>
<td>Mandatory</td>
<td>Object name and version number. For non-multi-version objects, set <code>version</code> to 0.</td>
</tr>
<tr>
<td>days</td>
<td>char *</td>
<td>Mandatory</td>
<td>Retention period of the restored object</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Mandatory or Optional</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>tier</td>
<td>obs_tier</td>
<td>Optional</td>
<td>Options for restoring: obs_tier.OBS_TIER_EXPEDITED, obs_tier.OBS_TIER_STANDARD, and obs_tier.OBS_TIER_BULK.</td>
</tr>
<tr>
<td>handler</td>
<td>obs_response_handler</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // Set response callback function.
    obs_response_handler handler =
        {
            &response_properties_callback, &response_complete_callback
        };
    // Restore 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 the days when the object is retained. The value range is 1 to 30.

6.4 Performing a Resumable Download

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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>key</td>
<td>char *</td>
<td>Mandatory</td>
<td>Object name</td>
</tr>
<tr>
<td>version_id</td>
<td>char *</td>
<td>Mandatory</td>
<td>Object version ID</td>
</tr>
<tr>
<td>download_file_config</td>
<td>obs_download_file_configuration *</td>
<td>Mandatory</td>
<td>For details about how to download files, see the obs_download_file_configuration member description in the following table.</td>
</tr>
</tbody>
</table>
### Field Descriptions

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_conditions</td>
<td>obs_get_conditions *</td>
<td>Mandatory</td>
<td>For details about how to set the object filter conditions and read range, see <a href="#">6.2 Performing a Conditioned Download</a>.</td>
</tr>
<tr>
<td>encryption_params</td>
<td>server_side_encryption_params *</td>
<td>Optional</td>
<td>Encryption setting of the uploaded object</td>
</tr>
<tr>
<td>handler</td>
<td>obs_download_file_response_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>download_file</td>
<td>char *</td>
<td>Mandatory</td>
<td>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.</td>
</tr>
<tr>
<td>part_size</td>
<td>uint64_t</td>
<td>Mandatory</td>
<td>Part size, in bytes. The value ranges from 5 MB (default) to 5 GB.</td>
</tr>
<tr>
<td>task_num</td>
<td>int</td>
<td>Mandatory</td>
<td>Maximum number of parts that can be concurrently downloaded. The default value is 1.</td>
</tr>
<tr>
<td>enable_check_point</td>
<td>int</td>
<td>Mandatory</td>
<td>Whether to enable the resumable upload mode. The default value is 0, which indicates that this mode is disabled.</td>
</tr>
<tr>
<td>check_point_file</td>
<td>char *</td>
<td>Mandatory</td>
<td>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.</td>
</tr>
</tbody>
</table>
Sample code:

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // Initialize getConditions.
    obs_get_conditions getConditions;
    memset_s(&getConditions, sizeof(obs_get_conditions), 0,
             sizeof(obs_get_conditions));
    // 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,
        &downloadFileResultCallback
    };
    download_file(&option, key, 0, &getConditions, 0, &downloadFileConfig,
                  &Handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("test download file successfully. \n");
    } else {
        printf("test download file failed(%s).\n",
               obs_get_status_name(ret_status));
    }
}
```

**NOTE**

- The API for resumable download, which is implemented based on 5.4 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.

### 6.5 Processing an Image

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

```c
static void test_get_object_image()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.hostname = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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.
    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 failed(%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.
7 Object Management

7.1 Obtaining Object Properties

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

Parameter Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>object_info</td>
<td>obs_object_info *</td>
<td>Mandatory</td>
<td>Object name and version number. For non-multi-version objects, set version to 0.</td>
</tr>
<tr>
<td>encryption_params</td>
<td>server_side_encryption_params *</td>
<td>Optional</td>
<td>Parameters related to server-side encryption</td>
</tr>
<tr>
<td>handler</td>
<td>obs_get_object_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

Sample Code

Sample code:

```c
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.hostName = "<your-endpoint>";
option.bucket_options.bucketName = "<Your bucketname>";
option.bucket_options.secretAccessKey = "<Your SK>";
// Object information
obs_object_info objectinfo;
memset(&objectinfo, 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, &objectinfo, 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));
}

7.2 Managing Object ACLs

Object ACLs, similar to bucket ACLs, support pre-defined access control policies and direct configuration. For details, see 4.5 Managing Bucket ACLs.

An object ACL can be configured in three modes:

1. Specify a pre-defined access control policy during object upload.
2. Call `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:

```c
static void test_put_object_acl()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.secretAccessKey = "<Your AK>";
    // 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;
```
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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>object_info</td>
<td>obs_object_info *</td>
<td>Mandatory</td>
<td>Object name and version number. For non-multi-version objects, set version to 0.</td>
</tr>
<tr>
<td>canned_acl</td>
<td>obs_canned_acl</td>
<td>Mandatory</td>
<td>For details about how to manage bucket access rights, see Table 4-1 in 4.5 Managing Bucket ACLs.</td>
</tr>
<tr>
<td>handler</td>
<td>obs_response_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

Sample code:

```c
void test_set_object_acl_byhead()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    // 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));
    }
}
```
option.bucket_options.hostName = "<your-endpoint>";
option.bucket_options.bucketName = "<Your bucketname>";
option.bucket_options.accessKeyId = "<Your AK>";
option.bucket_options.secretAccessKey = "<Your SK>";

// Set response callback function.
obs_response_handler response_handler =
{
    0, &response_complete_callback
};
obs_canned_acl canned_acl = OBS_CANNED_ACL_PUBLIC_READ_WRITE;
observerinfo = 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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>aclinfo</td>
<td>manager_acl_info *</td>
<td>Mandatory</td>
<td>Structure for managing ACL permission information</td>
</tr>
<tr>
<td>aclinfo-&gt;object_info</td>
<td>obs_object_info *</td>
<td>Mandatory</td>
<td>Object name and version number. For non-multi-version objects, set version to 0.</td>
</tr>
<tr>
<td>aclinfo-&gt;owner_id</td>
<td>char *</td>
<td>Optional</td>
<td>User account ID</td>
</tr>
<tr>
<td>aclinfo-&gt;acl_grant_count_return</td>
<td>int *</td>
<td>Mandatory</td>
<td>Pointer to the number of returned aclinfo-&gt;acl_grants</td>
</tr>
<tr>
<td>aclinfo-&gt;acl_grants</td>
<td>obs_acl_grant *</td>
<td>Mandatory</td>
<td>Pointer to the permission information structure. For details, see Table 4-2 in 4.5 Managing Bucket ACLs.</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Mandatory or Optional</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>aclinfo-&gt;object_delivered</td>
<td>obs_object_delivered</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>handler</td>
<td>obs_response_handler</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

Sample code:

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // Set response callback function.
    obs_response_handler response_handler =
        &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 Credential page of OBS Console.
Obtaining the Object ACL

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

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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);
}
```

7.3 Listing Objects

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

Parameter Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>prefix</td>
<td>char *</td>
<td>Optional</td>
<td>Name prefix that the objects to be listed must contain</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Mandatory or Optional</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>marker</td>
<td>char *</td>
<td>Optional</td>
<td>Object name to start with when listing objects in a bucket. All objects are listed in the lexicographical order</td>
</tr>
<tr>
<td>delimiter</td>
<td>char *</td>
<td>Optional</td>
<td>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.)</td>
</tr>
<tr>
<td>maxkeys</td>
<td>int</td>
<td>Mandatory</td>
<td>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.</td>
</tr>
<tr>
<td>handler</td>
<td>obs_list_objects_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

### Sample Code

```c
static void test_list_bucket_objects(char *bucket_name)
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";

    // 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.
}"
```
```c
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.
```

## 7.4 Deleting Objects

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

### Deleting a Single Object

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see <a href="#">3.3 Configuring option</a></td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>object_info</td>
<td>obs_object_info *</td>
<td>Mandatory</td>
<td>Object name and version number. For non-multi-version objects, set version to 0.</td>
</tr>
<tr>
<td>handler</td>
<td>obs_response_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

Sample code:

```c
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.hostname = "<your-endpoint>";
option.bucket_options.bucketName = "Your bucketname";
option.bucket_options.accessKeyId = "<Your AK>";
option.bucket_options.secretAccessKey = "<Your SK>";
// Set response callback function.
obs_response_handler responseHandler =
{
    &response_properties_callback,
    &response_complete_callback
};
// Delete an object.
delete_object(&option, &object_info, &responseHandler, &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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see <a href="#">3.3 Configuring option</a></td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>object_info</td>
<td>obs_object_info *</td>
<td>Mandatory</td>
<td>Name and version number of the object to be deleted. For non-multi-version objects, set version to 0.</td>
</tr>
<tr>
<td>delobj</td>
<td>obs_delete_object_info *</td>
<td>Mandatory</td>
<td>The number of objects to be deleted. Specifies the quiet mode.</td>
</tr>
</tbody>
</table>
### Field | Type | Mandatory or Optional | Description
--- | --- | --- | ---
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:

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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 failed(%s).\n",
        obs_get_status_name(ret_status));
    }
}
```
7.5 Copying an Object

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 is an Archive object, you must restore it before copying it.

## Copying an Object in Simple Mode

The parameters are as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>request for the</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td></td>
<td>context of the bucket,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>see 3.3 Configuring</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>option</td>
<td></td>
<td></td>
</tr>
<tr>
<td>key</td>
<td>char *</td>
<td>Mandatory</td>
<td>Object name</td>
</tr>
<tr>
<td>version_id</td>
<td>char *</td>
<td>Optional</td>
<td>Object version ID</td>
</tr>
<tr>
<td>object_info</td>
<td>obs_copy_destination</td>
<td>Mandatory</td>
<td>Indicates the information of the copied object.</td>
</tr>
<tr>
<td></td>
<td>_object_info *</td>
<td></td>
<td></td>
</tr>
<tr>
<td>object_info-&gt;destination_bucke</td>
<td>char *</td>
<td>Mandatory</td>
<td>Bucket where the target object is located</td>
</tr>
<tr>
<td>t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>object_info-&gt;destination_key</td>
<td>char *</td>
<td>Mandatory</td>
<td>Target object name</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>object_info-&gt;last_modified_return</td>
<td>int64_t *</td>
<td>Mandatory</td>
<td>Latest time when the object was modified</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>object_info-&gt;etag_return_size</td>
<td>int</td>
<td>Mandatory</td>
<td>Cache size of ETag</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>object_info-&gt;etag_return</td>
<td>char *</td>
<td>Mandatory</td>
<td>Cache of ETag</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Field | Type | Mandatory or Optional | Description
--- | --- | --- | ---
**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:**

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>if_modified_since</td>
<td>Copies the source object if it is changed after the time specified by this parameter; otherwise, an exception is thrown.</td>
</tr>
<tr>
<td>if_not_modified_since</td>
<td>Copies the source object if it is changed before the time specified by this parameter; otherwise, an exception is thrown.</td>
</tr>
<tr>
<td>if_match_etag</td>
<td>Copies the source object if its ETag is the same as the one specified by this parameter; otherwise, an exception is thrown.</td>
</tr>
<tr>
<td>if_not_match_etag</td>
<td>Copies the source object if its ETag is different from the one specified by this parameter; otherwise, an exception is thrown.</td>
</tr>
</tbody>
</table>

**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:

```java
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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>";
```
Modifying an Object ACL

Sample code:

```c
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.
    init_obs_options(&option);
    option.bucket_options.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // Set the destination object information.
    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 Temporarily Authorized Request

8.1 What Is a Temporarily Authorized Request

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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Structure in SDK</th>
</tr>
</thead>
<tbody>
<tr>
<td>expires</td>
<td>Validity period of the generated temporary URL</td>
<td>obs_options. temp_auth_configure</td>
</tr>
<tr>
<td>temp_auth_callback</td>
<td>The callback function used to return the generated temporary URL</td>
<td></td>
</tr>
<tr>
<td>callback_data</td>
<td>Callback data</td>
<td></td>
</tr>
</tbody>
</table>

8.2 Temporarily Authorized Request

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

8.2.1 Creating a Bucket

```c
static void test_create_bucket_auth()
{
    // Create and initialize option.
    obs_options option;
```
8.2.2 Uploading an Object

```c
static void test_put_object_auth()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    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 responseHandler =
    {
        &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("create bucket successfully. \n");
    } else {
        printf("create bucket failed(%s)\n", obs_get_status_name(ret_status));
    }
}
```
8.2.3 Downloading an Object

```c
static void test_get_object_auth()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    temp_auth_configure tempauth;
    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;
    // 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("get object successfully. \n");
    }
    else {
        printf("get object failed(%s).\n", obs_get_status_name(data.ret_status));
    }
}
```
8.2.4 Listing Objects

static void test_list_object_auth()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    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("list bucket objects successfully. \n");
    } else {
        printf("list bucket objects failed\n",
            obs_get_status_name(data.ret_status));
    }
}

8.2.5 Deleting an Object

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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    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;
memcpy(&object_info,0,sizeof(obs_object_info));
object_info.key = key;
// Assign a value to the callback function.
obs_response_handler responseHandler =
{
  &response_properties_callback,
  &response_complete_callback
};
// Delete an object.
delete_object(&option,&object_info,&responseHandler, &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));
}
9 Versioning Management

9.1 Versioning Overview

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

For details, see Versioning.

9.2 Setting Versioning Status for a Bucket

You can call `set_bucket_version_configuration` to set the versioning status for a bucket.

OBS supports two versioning statuses.
<table>
<thead>
<tr>
<th>Versioning Status</th>
<th>Description</th>
<th>Value in OBS C SDK</th>
</tr>
</thead>
</table>
| 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 |
<table>
<thead>
<tr>
<th>Versioning Status</th>
<th>Description</th>
<th>Value in OBS C SDK</th>
</tr>
</thead>
</table>
| 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**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see <a href="#">3.3 Configuring option</a></td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>handler</td>
<td>obs_response_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
</tbody>
</table>
### 9.3 Viewing Versioning Status of a Bucket

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

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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));
    }
}
```

#### Parameter Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>status_return_size</td>
<td>int</td>
<td>Mandatory</td>
<td>Size of the versioning status cache</td>
</tr>
</tbody>
</table>
9.4 Obtaining a Versioning Object

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:

```c
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.hostName = "<your-endpoint>";
  // Set response callback function.
  obs_response_handler response_handler =
  {
    &response_properties_callback,
    &response_complete_callback
  };
  // Obtain bucket versioning status.
  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));
  }
}
```
NOTE

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

9.5 Copying a Versioning Object

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

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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");
    }
}
```
9.6 Restoring a Versioning Archive Object

You can call `restore_object` to pass the version ID (`obs_object_info.version_id`) to restore a versioning Archive object. Sample code is as follows:

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    obs_tier tier = OBS_TIER_EXPEDITED;
    // Set response callback function.
    obs_response_handler handler =
    {
        &response_properties_callback,
        &response_complete_callback
    };
    // Restore a versioning Archive object.
    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));
    }
}
```

9.7 Listing Versioning Objects

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

The following table describes the parameters involved in this API.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Mandatory or Optional</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>prefix</td>
<td>char *</td>
<td>Optional</td>
<td>Name prefix that the objects to be listed must contain</td>
</tr>
<tr>
<td>key_marker</td>
<td>char *</td>
<td>Optional</td>
<td>Versioning object name to start with when listing versioning objects in a bucket. All versioning objects are listed in the lexicographical order.</td>
</tr>
<tr>
<td>delimiter</td>
<td>char *</td>
<td>Optional</td>
<td>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.)</td>
</tr>
<tr>
<td>version_id_marker</td>
<td>char *</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>maxkeys</td>
<td>int</td>
<td>Mandatory</td>
<td>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.</td>
</tr>
<tr>
<td>handler</td>
<td>obs_list_versions_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

**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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // Set response callback function.
    obs_list_versions_handler list_versions_handler =
        {
            &response_properties_callback,
            &list_versions_complete_callback
        };
    // Create and Initialize callback data.
    list_versions_callback_data data;
    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 versioning objects, you can list objects in paging mode by
    specifying the object prefix using prefix and specifying the number using
    maxkeys. delimiter specifies the character in the group listing. To group in
    folders, set delimiter to forward slash (/), use key Marker to specify the start
    position of the versioning, and use version_id_marker to specify the version IDs.
    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.

### 9.8 Setting or Obtaining a Versioning Object ACL

#### 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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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 needed in the ACL indicates the account ID, which can be viewed on the My Credential page of OBS Console.

Obtaining a Versioning Object ACL

You can call 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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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
", 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);

9.9 Deleting Versioning Objects

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:

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // Set response callback function.
    obs_response_handler responseHandler = {
        &response_properties_callback,
        &response_complete_callback
    };
    // Delete an object.
    delete_object(&option, &object_info, &responseHandler, &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:

```c
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);
```
```c
option.bucket_options.hostName = "<your-endpoint>";
option.bucket_options.bucketName = "<Your bucketname>";
option.bucket_options.accessKeyId = "<Your AK>";
option.bucket_options.secretAccessKey = "<Your SK>";
// 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 failed(%s).\n",
obs_get_status_name(ret_status));
}
```
10 Lifecycle Management

10.1 Lifecycle Management Overview

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

- Rule ID, which uniquely identifies the rule
- Prefix of objects that are under the control of this rule
- Transition policy of an object of the latest version, which can be specified in either mode:
  a. How many days after the object is created
  b. Transition date
- Expiration time of an object of the latest version, which can be specified in either mode:
  a. How many days after the object is created
  b. Expiration date
- Transition 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.
10.2 Setting Lifecycle Rules

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

Parameter Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see <a href="#">3.3 Configuring option</a></td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>bucket_lifecycle_conf</td>
<td><code>obs_lifecycle_conf</code>*</td>
<td>Mandatory</td>
<td>For details about the bucket lifecycle configuration, see the following table.</td>
</tr>
<tr>
<td>blcc_number</td>
<td>unsigned int</td>
<td>Mandatory</td>
<td>Number of array members in the array <code>bucket_lifecycle_conf</code>.</td>
</tr>
<tr>
<td>handler</td>
<td><code>obs_response_handler</code>*</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td><code>void</code>*</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

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

---

[3.3 Configuring option](#)
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>const char *</td>
<td></td>
<td>If there is no days and no transition, noncurrent_version_days, noncurrent_version_transition is available, 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).</td>
</tr>
<tr>
<td>days</td>
<td>const char *</td>
<td></td>
<td>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).</td>
</tr>
<tr>
<td>id</td>
<td>const char *</td>
<td>Optional</td>
<td>Unique identifier of a rule. The value can contain a maximum of 255 characters.</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Mandatory or Optional</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------</td>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>prefix</td>
<td>const char *</td>
<td>Mandatory</td>
<td>Object name prefix identifying one or more objects to which the rule applies.</td>
</tr>
<tr>
<td>status</td>
<td>const char *</td>
<td>Mandatory</td>
<td>Indicates whether the rule is enabled.</td>
</tr>
<tr>
<td>noncurrent_version_days</td>
<td>const char *</td>
<td>Optional</td>
<td>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).</td>
</tr>
<tr>
<td>transition_num</td>
<td>unsigned int</td>
<td></td>
<td>If <strong>transition</strong> is not empty, this parameter is mandatory. Number of array members in the array <strong>transition</strong>.</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Mandatory or Optional</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>transition</td>
<td>obs_lifecycle_transition *</td>
<td>If there are no date, days, noncurrent_version_transition, or noncurrent_version_days, this parameter is mandatory.</td>
<td>Indicates the transition time and the object storage class after transition (valid only for the latest object version).</td>
</tr>
<tr>
<td>transition-&gt;date</td>
<td>const char *</td>
<td></td>
<td>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).</td>
</tr>
<tr>
<td>transition-&gt;days</td>
<td>const char *</td>
<td></td>
<td>Number of days when the transition rule takes effect after the object creation (only applicable to latest versions of objects).</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Mandatory or Optional</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>transition- &gt;storage_class</td>
<td>obs_storage_class</td>
<td>If there is transition, this parameter is mandatory.</td>
<td>The storage class to which the latest version object is modified.</td>
</tr>
<tr>
<td>noncurrent_version_transition_num</td>
<td>unsigned int</td>
<td>If obs_lifecycle_noncurrent_transition is not empty, this parameter is mandatory.</td>
<td>Number of array members in the array noncurrent_version_transition.</td>
</tr>
<tr>
<td>noncurrent_version_transition</td>
<td>obs_lifecycle_noncurrent_transition *</td>
<td>If there are no date, days, transition, or noncurrent_version_days, this parameter is mandatory.</td>
<td>Transition time of noncurrent object versions and the object storage class after transition.</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Mandatory or Optional</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------</td>
<td>-----------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>noncurrent_version_transition-&gt;noncurrent_version_days</td>
<td>const char *</td>
<td>If there is noncurrent_version_transition, this parameter is mandatory.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>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).</td>
</tr>
<tr>
<td>noncurrent_version_transition-&gt;storage_class</td>
<td>obs_storage_class</td>
<td>If there is noncurrent_version_transition, this parameter is mandatory.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The storage class to which the noncurrent version object is modified.</td>
</tr>
</tbody>
</table>

Setting an Object Transition Policy

Sample code:

```c
static void test_set_bucket_lifecycle_configuration1()
{
    obs_options option;
    obs_status  ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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";
}```
Setting an Object Expiration Time

Sample code:

```c
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;
    option.bucket_options.access_key = ACCESS_KEY_ID;
    option.bucket_options.secret_access_key = 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 prefix will be transited 30 days after creation.
    transition.days = "30";
    // Specify that objects whose names contain the prefix will be transited after changing into noncurrent versions for 20 days.
    noncurrent_version_days = "20";
    // The lifecycle rule takes effect.
    set_bucket_lifecycle_configuration(&option, &bucket_lifecycle_conf, 1,
                                        &response_handler, &ret_status);
}
```

// Specify that objects whose names contain the prefix will expire 10 days after creation.
bucket_lifecycle_conf.days = "10";
obslifecycle_transition transition;
memset(&transition, 0, sizeof(obs_lifecycle_transition));
// Specify that objects whose names contain the prefix will be transited 30 days after creation.
transition.days = "30";
// Specify that objects whose names contain the prefix will be transited after changing into noncurrent versions for 20 days.
noncurrent_version_days = "20";
// The lifecycle rule takes effect.
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));
}

10.3 Viewing Lifecycle Rules

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

Parameter Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>handler</td>
<td>obs_lifecycle_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

Sample Code

```c
static void test_get_bucket_lifecycle_configuration()
{
    obs_options option;
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    init_obs_options(&option);
    option.bucket_options.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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 failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```
10.4 Deleting Lifecycle Rules

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

### Parameter Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>handler</td>
<td>obs_response_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

### Sample Code

```c
static void test_delete_bucket_lifecycle_configuration()
{
    obs_options option;
    obs_status ret_status = OBS_STATUS_BUTT;
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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));
    }
}
```
11 Cross-Origin Resource Sharing (CORS)

11.1 CORS Overview

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

For more information, see CORS.

11.2 Setting CORS Rules

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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>obs_cors_conf_info</td>
<td>obs_bucket_cors_conf</td>
<td>Mandatory</td>
<td>For details about CORS rules, see the following table.</td>
</tr>
<tr>
<td>conf_num</td>
<td>unsigned int</td>
<td>Mandatory</td>
<td>Number of array members in the array <code>obs_cors_conf_info</code>.</td>
</tr>
<tr>
<td>handler</td>
<td>obs_response_handler</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
</tbody>
</table>
The following table describes the CORS rule structure `obs_bucket_cors_conf`.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>const char *</td>
<td>Optional</td>
<td>Object names in a bucket</td>
</tr>
<tr>
<td>allowed_method</td>
<td>const char **</td>
<td>Mandatory</td>
<td>Method allowed by a CORS rule</td>
</tr>
<tr>
<td>allowed_method_number</td>
<td>unsigned int</td>
<td>Mandatory</td>
<td>Number of <code>allowed_method</code></td>
</tr>
<tr>
<td>allowed_origin</td>
<td>const char **</td>
<td>Mandatory</td>
<td>Indicates an origin that is allowed by a CORS rule. It is a character string and can contain a wildcard (<em>), and allows one wildcard character (</em>) at most.</td>
</tr>
<tr>
<td>allowed_origin_number</td>
<td>unsigned int</td>
<td>Mandatory</td>
<td>Number of <code>allowed_origin</code></td>
</tr>
<tr>
<td>allowed_header</td>
<td>const char **</td>
<td>Optional</td>
<td>Indicates which headers are allowed in a PUT Bucket CORS request via the <code>Access-Control-Request-Headers</code>. If a request contains <code>Access-Control-Request-Headers</code>, only a CORS request that matches the configuration of <code>allowed_header</code> is considered as a valid request. Each <code>allowed_header</code> can contain at most one wildcard (*) and cannot contain spaces.</td>
</tr>
<tr>
<td>allowed_header_number</td>
<td>unsigned int</td>
<td>Optional</td>
<td>Number of <code>allowed_header</code></td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Mandatory or Optional</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>max_age_seconds</td>
<td>const char *</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>expose_header</td>
<td>const char **</td>
<td>Optional</td>
<td>Indicates a supplemented header in CORS responses. The header provides additional information for clients. It cannot contain spaces.</td>
</tr>
<tr>
<td>expose_header_number</td>
<td>unsigned int</td>
<td>Optional</td>
<td>Number of expose_header.</td>
</tr>
</tbody>
</table>

Sample Code

```c
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>";
    option->bucket_options.accessKeyId = "<Your AK>";
    option->bucket_options.secretAccessKey = "<Your SK>";
    // 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.
    // 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 failed(%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.

11.3 Viewing CORS Rules

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

Parameter Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>handler</td>
<td>obs_cors_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

Sample Code

```c
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>";
    option->bucket_options.accessKeyId = "<Your AK>";
    option->bucket_options.secretAccessKey = "<Your SK>";
    // Set callback function.
    obs_cors_handler cors_handler_info =
    {
        &response_properties_callback, &response_complete_callback,
        &get_cors_info_callback
    }
```
11.4 Deleting CORS Rules

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

**Parameter Description**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>handler</td>
<td>obs_response_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

```c
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>";
    option->bucket_options.secretKeyId = "<Your AK>";
    option->bucket_options.secretAccessKey = "<Your SK>";
    // 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 failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```
12 Setting Access Logging

12.1 Logging Overview

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

For more information, see Logging.

12.2 Enabling Bucket Logging

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>target_bucket</td>
<td>char *</td>
<td>Mandatory</td>
<td>When enabling the logging function, the owner of the bucket being logged can specify a target bucket to store the generated log files. You can deliver logs to any bucket that you own. User groups that deliver logs to the target bucket must have the WRITE and READ_ACP permissions. 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.</td>
</tr>
<tr>
<td>target_prefix</td>
<td>char *</td>
<td>Mandatory</td>
<td>You can specify a prefix using this element so that log files are named with this prefix.</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Mandatory or Optional</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------</td>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>agency</td>
<td>char *</td>
<td>Mandatory</td>
<td>This parameter is mandatory when enabling bucket logging. Do not select this parameter when disabling bucket logging.</td>
</tr>
<tr>
<td>acl_group</td>
<td>obs_acl_group *</td>
<td>Optional</td>
<td>Name of the agency created by the owner of the logging bucket for uploading log files by OBS.</td>
</tr>
<tr>
<td>acl_group-&gt;acl_grant_count</td>
<td>int</td>
<td>Optional</td>
<td>Structure of the permission information group</td>
</tr>
<tr>
<td>acl_group-&gt;acl_grants</td>
<td>obs_acl_grant *</td>
<td>Optional</td>
<td>The number of returned obs_acl_grant.</td>
</tr>
<tr>
<td>handler</td>
<td>obs_response_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

**Sample Code**

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
}
```
Setting ACLs for Objects to Be Logged

Sample code:

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // Set response callback function.
    obs_response_handler response_handler =
    { 0,
       &response_complete_callback
    };
    int aclGrantCount = 2;
    obs_acl_group acl_group;
    g.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");
    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));
    }
}
```
12.3 Viewing Bucket Logging Configuration

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

Parameter Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>logging_message_</td>
<td>bucket_logging_message</td>
<td>Mandatory</td>
<td>Current bucket logging management configuration. For details about bucket_logging_message, see the following table.</td>
</tr>
<tr>
<td>data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>handler</td>
<td>obs_response_handler</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target_bucket</td>
<td>char *</td>
<td>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.</td>
</tr>
<tr>
<td>target_bucket_size</td>
<td>int</td>
<td>Total size of <code>target_bucket</code></td>
</tr>
<tr>
<td>target_prefix</td>
<td>char *</td>
<td>You can specify a prefix using this element so that log files are named with this prefix.</td>
</tr>
<tr>
<td>target_prefix_size</td>
<td>int</td>
<td>Total size of <code>target_prefix</code></td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>acl_grants</td>
<td>obs_acl_grant *</td>
<td>Pointer to the permission information structure</td>
</tr>
<tr>
<td>acl_grant_count</td>
<td>int *</td>
<td>Pointer to the number of returned acl_grants</td>
</tr>
<tr>
<td>agency</td>
<td>char *</td>
<td>Name of the agency created by the owner of the logging bucket for uploading log files by OBS</td>
</tr>
<tr>
<td>agency_size</td>
<td>int</td>
<td>Total size of agency</td>
</tr>
</tbody>
</table>

**Sample Code**

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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));
    }
}
```
12.4 Disabling Bucket Logging

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

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // Set response callback function.
    obs_response_handler response_handler =
        { 0, &response_complete_callback };
    // Disable bucket logging.
    set_bucket_logging_configuration_ob(&option, 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));
    }
}
```
13 Static Website Hosting

13.1 Static Website Hosting Overview

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.

13.2 Website File Hosting

You can perform the following to implement website file hosting:

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

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

**Step 3** Access the object using a browser.

--- End

Sample code:

```
static void test_put_object() {
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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));
}

13.3 Setting Website Hosting

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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>set_bucket_redirect_all</td>
<td>obs_set_bucket_redirect_all_conf*</td>
<td>Mandatory</td>
<td>Indicates the redirection configuration.</td>
</tr>
<tr>
<td>set_bucket_redirect_all-&gt;host_name</td>
<td>const char *</td>
<td>Mandatory</td>
<td>Indicates the name of the host where requests will be redirected.</td>
</tr>
<tr>
<td>set_bucket_redirect_all-&gt;protocol</td>
<td>const char *</td>
<td>Optional</td>
<td>The HTTP or HTTPS protocol used in redirecting requests. The default protocol is HTTP.</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Mandatory or Optional</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>-----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>set_bucket_website_conf</td>
<td>obs_set_bucket_website_conf *</td>
<td>Mandatory</td>
<td>Describes the configuration of the website element in the redirection rules.</td>
</tr>
<tr>
<td>set_bucket_website_conf-&gt;suffix</td>
<td>const char *</td>
<td>Mandatory</td>
<td>Suffix that is appended to a request initiated for a directory on the website endpoint. For example, if the suffix is <code>index.html</code> and you request for <code>samplebucket/images/</code>, the data that is returned will be the object with the key name <code>images/index.html</code> in the samplebucket bucket. Suffix cannot be empty or contain slashes (/).</td>
</tr>
<tr>
<td>set_bucket_website_conf-&gt;key</td>
<td>const char *</td>
<td>Optional</td>
<td>Indicates the object name that is used when a 4XX error occurs. This element identifies the page that is returned when a 4XX error occurs.</td>
</tr>
<tr>
<td>set_bucket_website_conf-&gt;routingrule_info</td>
<td>bucket_website_routingrule *</td>
<td>Optional</td>
<td>For details about redirection rules, see the following table.</td>
</tr>
<tr>
<td>set_bucket_website_conf-&gt;routingrule_count</td>
<td>int</td>
<td>Optional</td>
<td>Total size of <code>set_bucket_website_conf.routingrule_info</code></td>
</tr>
<tr>
<td>handler</td>
<td>obs_response_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key_prefix_equals</td>
<td>const char *</td>
<td>Optional</td>
<td>Indicates the object name prefix when the redirection is applied.</td>
</tr>
</tbody>
</table>
Field | Type          | Mandatory or Optional | Description                                
--- | -------------- |---------------------- |--------------------------------------------
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.
http_redirect_code | const char * | Optional | Indicates the HTTP status code returned after the redirection request.

```c
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.hostname = "<your-endpoint>";
    option.bucket_options.bucketname = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    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].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].http_errorcode_returned_equals="404”;
    temp[1].http_redirect_code = NULL;
    temp[1].host_name = "www.xxx.com";
    temp[1].protocol = "http";
    // Set response callback function.
```
```c
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:**

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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.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));
    }
}
```

### 13.4 Viewing Website Hosting Settings

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>handler</td>
<td>obs_get_bucket_websiteconf_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>handler-&gt;response_handler</td>
<td>obs_response_handler</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>handler-&gt;get_bucket_websiteconf_callback</td>
<td>obs_get_bucket_websiteconf_callback *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

### Sample Code

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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\n",
        obs_get_status_name(ret_status));
    }
}
```

### 13.5 Deleting Hosting Settings

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>handler</td>
<td>obs_response_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

Sample Code

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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));
    }
}
```
14 Tag Management

14.1 Tagging Overview

Tags are used to identify and classify OBS buckets.

For more information, see Tags.

14.2 Setting Bucket Tags

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

Parameter Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>tagging_list</td>
<td>obs_name_value *</td>
<td>Mandatory</td>
<td>Tag list</td>
</tr>
<tr>
<td>number</td>
<td>unsigned int</td>
<td>Mandatory</td>
<td>Number of tags</td>
</tr>
<tr>
<td>handler</td>
<td>obs_response_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

The following table describes the tag list structure `obs_name_value`. 
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>char *</td>
<td>Tag key</td>
</tr>
<tr>
<td>value</td>
<td>char *</td>
<td>Tag value</td>
</tr>
</tbody>
</table>

Sample Code

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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.

### 14.3 Viewing Bucket Tags

You can call `get_bucket_tagging` to view bucket tags.
### Parameter Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Request for the context of the bucket, see 3.3 <strong>Configuring option</strong></td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>handler</td>
<td>obs_get_bucket_tagging_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>response_handler</td>
<td>obs_response_handler</td>
<td>Callback function handler for response</td>
</tr>
<tr>
<td>get_bucket_tagging_callb</td>
<td>obs_get_bucket_tagging_callee *</td>
<td>Callback function for obtaining bucket tags</td>
</tr>
</tbody>
</table>

### Sample Code

```c
static void test_get_bucket_tagging()
{
    // Create and initialize option.
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";

    // 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\n",
            obs_get_status_name(tagging_info.ret_status));
    }
}
```
14.4 Deleting Bucket Tags

You can call `delete_bucket_tagging` to delete bucket tags.

Parameter Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Request for the context of the bucket, see <strong>3.3 Configuring option</strong></td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>handler</td>
<td>obs_response_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

Sample Code

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    // 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));
    }
}
```
15 Event Notification

15.1 Event Notification Overview

OBS provides the event notification function. When this function is enabled, you will be notified of all specified operations on the bucket through simple message notification (SMN). Event notification contains the following configurations:

- Unique ID of each event notification. If the user does not specify an ID, OBS will automatically assign one.
- URN of the event notification topic. After detecting a specific event, OBS sends a message to the topic.
- List of event types that need to be notified.
- Filtering rules. Objects can be filtered based on the prefixes or suffixes.

For more information, see Event Notification.

15.2 Setting Event Notification

You can call set_notification_configuration to set event notification for a bucket.

Parameter Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Request for the context of the bucket, see 3.3 Configuring option</td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>notification_conf</td>
<td>obs_smn_notification_configuration*</td>
<td>Mandatory</td>
<td>Indicates the container for configuring the event notification.</td>
</tr>
</tbody>
</table>
The following table describes the parameter **obs_smn_notification_configuration**.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic_conf</td>
<td>obs_smn_topic_configuration*</td>
<td></td>
<td>Container for configuring the event notification topic</td>
</tr>
<tr>
<td>topic_conf_num</td>
<td>unsigned int</td>
<td></td>
<td>Number of <strong>topic_conf</strong></td>
</tr>
</tbody>
</table>

The following table describes the parameter **obs_smn_topic_configuration**.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic</td>
<td>char *</td>
<td>URN of the event notification topic. After detecting a specific event in the bucket, OBS sends a message to the topic.</td>
</tr>
<tr>
<td>id</td>
<td>char *</td>
<td>Unique ID of each event notification. If the user does not specify an ID, the system assigns an ID automatically.</td>
</tr>
<tr>
<td>filter_rule</td>
<td>obs_smn_filter_rule *</td>
<td>Container that defines key-value pairs of filter rules</td>
</tr>
<tr>
<td>filter_rule_num</td>
<td>unsigned int</td>
<td>Number of <strong>filter_rule</strong></td>
</tr>
<tr>
<td>event</td>
<td>obs_smn_event_enum*</td>
<td>Type of events that need to be notified</td>
</tr>
<tr>
<td>event_num</td>
<td>unsigned int</td>
<td>Number of event types</td>
</tr>
</tbody>
</table>

**Sample Code**

```c
static void test_set_notification_configuration()
{
    obs_options option;
    obs_status  ret_status = OBS_STATUS_BUTT;
    obs_smn_notification_configuration notification_conf;
    obs_smn_topic_configuration topic_conf;
    obs_smn_event_enum topic1_event[2];
```
```c
obs_smn_filter_rule filter_rule;
// Set option.
init_obs_options(&option);
option->bucket_options.hostName = "<your-endpoint>";
option->bucket_options.bucketName = "<Your bucketname>";
option->bucket_options.accessKeyId = "<Your AK>";
option->bucket_options.secretAccessKey = "<Your SK>";
// Set callback function.
obs_response_handler response_handler = {
    NULL, &response_complete_callback
};
// Set the unique ID of the notification configuration.
topic_conf.id = "Id001";
// Set the URN of the event notification topic.
topic_conf.topic = "urn:smn:southchina:ea79855fbe0642718cb4df1551c3cb4e:test_cwx298983";
// Set notification operations
topic_conf.event = topic1_event;
topic_conf.event[0] = SMN_EVENT_OBJECT_CREATED_ALL;
topic_conf.event[1] = SMN_EVENT_OBJECT_CREATED_POST;
topic_conf.event_num = 2;
// Set filtering rules for notification objects.
filter_rule.name = OBS_SMN_FILTER_PREFIX;
filter_rule.value = "aaa";
topic_conf.filter_rule = &filter_rule;
topic_conf.filter_rule_num = 1;
memset(&notification_conf, 0, sizeof(obs_smn_notification_configuration));
notification_conf.topic_conf = &topic_conf;
notification_conf.topic_conf_num = 1;
set_notification_configuration(&option, &notification_conf,
    &response_handler, &ret_status);
if (OBS_STATUS_OK == ret_status) {
    printf("set_notification_configuration success.\n");
} else {
    printf("set_notification_configuration failed(%s).\n",
        obs_get_status_name(ret_status));
}
```

### 15.3 Viewing Event Notification Settings

You can call `get_notification_configuration` to view event notification for a bucket.

#### Parameter Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Request for the context of the bucket, see <a href="#">3.3 Configuring option</a></td>
<td>Mandatory</td>
<td>Bucket parameter</td>
</tr>
<tr>
<td>handler</td>
<td>obs_smn_handler *</td>
<td>Mandatory</td>
<td>Callback function</td>
</tr>
<tr>
<td>callback_data</td>
<td>void *</td>
<td>Optional</td>
<td>Callback data</td>
</tr>
</tbody>
</table>

The following table describes the callback function types of `obs_smn_handler`.
### 15 Event Notification

#### 15.4 Disabling Event Notification

To disable event notification on buckets is to call `set_notification_configuration()` to clear all event notification settings. Sample code is as follows:

```c
static void test_close_notification_configuration()
{
    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>";
    option->bucket_options.accessKeyId = "<Your AK>";
    option->bucket_options.secretAccessKey = "<Your SK>";
    // Set callback function.
    obs_smn_notification_configuration notification_conf =
        {
            NULL, &response_complete_callback
        };
    memset(&notification_conf, 0, sizeof(obs_smn_notification_configuration));
    set_notification_configuration(&option, &notification_conf,
        &response_handler, &ret_status);
}
```
if (OBS_STATUS_OK == ret_status) {
    printf("close_notification_configuration success.\n");
} else {
    printf("close_notification_configuration failed(%s).\n",
            obs_get_status_name(ret_status));
}
16 Server-Side Encryption

16.1 Server-Side Encryption Overview

OBS supports server-side encryption.

For more information, see Server-Side Encryption.

16.2 Encryption Description

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

<table>
<thead>
<tr>
<th>API Method in OBS C SDK</th>
<th>Description</th>
<th>Supported Encryption Type</th>
<th>Transmission Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>put_object</td>
<td>Sets the encryption algorithm and key during object upload to enable server-side encryption.</td>
<td>SSE-KMS SSE-C</td>
<td>HTTPS</td>
</tr>
<tr>
<td>get_object</td>
<td>Sets the decryption algorithm and key during object download to decrypt the object.</td>
<td>SSE-KMS SSE-C</td>
<td>HTTPS</td>
</tr>
<tr>
<td>API Method in OBS C SDK</td>
<td>Description</td>
<td>Supported Encryption Type</td>
<td>Transmission Protocol</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------</td>
<td>--------------------------</td>
<td>-----------------------</td>
</tr>
</tbody>
</table>
| 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 |
<table>
<thead>
<tr>
<th>API Method in OBS C SDK</th>
<th>Description</th>
<th>Supported Encryption Type</th>
<th>Transmission Protocol</th>
</tr>
</thead>
</table>
| 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`.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| encryption_type            | obs_encryption_type | Encryption mode:  
                                 **OBS_ENCRYPTION_KMS**: SE-KMS;  
                                 **OBS_ENCRYPTION_SSE_C**: SSE-C |
<p>| 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. |
| ssec_customer_key          | char *       | Indicates the key used to encrypt an object. The parameter is used in SSE-C mode. |</p>
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>des_ssec_customer_algorithm</td>
<td>char *</td>
<td>Indicates the algorithm used to decrypt a source object. The parameter is used in SSE-C mode.</td>
</tr>
<tr>
<td>des_ssec_customer_key</td>
<td>char *</td>
<td>Indicates the key used to decrypt a source object. The parameter is used in SSE-C mode.</td>
</tr>
</tbody>
</table>

### 16.3 Example of Encryption

#### Encrypting an Object to Be Uploaded

Sample code:

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    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.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.use_ssec = '1';
    encryption_params.ssec_customer_algorithm = "AES256";
    encryption_params.ssec_customer_key = "K7QkYpBkM5+hcs27fs9kUnvAobcnLht/iCB2o/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");
    }
}
```
Decrypted a To-Be-Download Object

Sample code:

```c
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.hostName = "<your-endpoint>";
    option.bucket_options.bucketName = "<Your bucketname>";
    option.bucket_options.accessKeyId = "<Your AK>";
    option.bucket_options.secretAccessKey = "<Your SK>";
    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+hcs27fsNkUnNvaobcnLht/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 failed(%s).\n",
               obs_get_status_name(data.ret_status));
    }
    fclose(data.outfile);
}
```
17.1 OBS Server-Side Error Codes

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

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
<th>HTTP Status Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccessDenied</td>
<td>Access denied.</td>
<td>403 Forbidden</td>
</tr>
<tr>
<td>AccessForbidden</td>
<td>Insufficient permission.</td>
<td>403 Forbidden</td>
</tr>
<tr>
<td>AccountProblem</td>
<td>Your account encounters a problem that prevents the operation from completing. The account may be expired or frozen.</td>
<td>403 Forbidden</td>
</tr>
<tr>
<td>AllAccessDisabled</td>
<td>The user has no permission to perform a specific operation.</td>
<td>403 Forbidden</td>
</tr>
<tr>
<td>AmbiguousGrantByEmailAddress</td>
<td>Multiple accounts share one email address.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>BadDigest</td>
<td>The specified value of Content-MD5 does not match the value received by OBS.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>BadDomainName</td>
<td>Invalid domain name.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>BadRequest</td>
<td>Invalid request parameters.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>BucketAlreadyExists</td>
<td>The requested bucket name already exists. The bucket namespace is shared by all users of OBS. Select another name and retry.</td>
<td>409 Conflict</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
<td>HTTP Status Code</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>BucketAlreadyOwnedByYou</td>
<td>Your previous request for creating the named bucket succeeded and you already own it.</td>
<td>409 Conflict</td>
</tr>
<tr>
<td>BucketNotEmpty</td>
<td>The bucket that you tried to delete is not empty.</td>
<td>409 Conflict</td>
</tr>
<tr>
<td>CredentialsNotSupported</td>
<td>This request does not support security credentials.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>CustomDomainAlreadyExist</td>
<td>The configured domain already exists.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>CustomDomainNotExist</td>
<td>The domain to be operated does not exist.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>DeregisterUserId</td>
<td>The user has been deregistered.</td>
<td>403 Forbidden</td>
</tr>
<tr>
<td>EntityTooSmall</td>
<td>The size of the object to be uploaded is smaller than the lower limit.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>EntityTooLarge</td>
<td>The size of the object to be uploaded exceeds the upper limit.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>FrozenUserId</td>
<td>The user has been frozen.</td>
<td>403 Forbidden</td>
</tr>
<tr>
<td>IllegalVersioningConfigurationException</td>
<td>The configuration of versioning in the request is invalid.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>IllegalLocationConstraintException</td>
<td>The configured region limitation is inconsistent with the region where it resides.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>InArrearOrInsufficientBalance</td>
<td>The user has no permission to perform some operations due to being in arrears or insufficient funds.</td>
<td>403 Forbidden</td>
</tr>
<tr>
<td>IncompleteBody</td>
<td>Incomplete request body.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>IncorrectNumberOfFilesInPostRequest</td>
<td>Each POST request must contain one file to be uploaded.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>InlineDataTooLarge</td>
<td>The size of inline data exceeds the upper limit.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>InsufficientStorageSpace</td>
<td>Insufficient storage space.</td>
<td>403 Forbidden</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
<td>HTTP Status Code</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>InternalError</td>
<td>An internal error occurs. Retry later.</td>
<td>500 Internal Server Error</td>
</tr>
<tr>
<td>InvalidAccessKeyId</td>
<td>The access key ID provided by the customer does not exist in the system.</td>
<td>403 Forbidden</td>
</tr>
<tr>
<td>InvalidAddressingHeader</td>
<td>The anonymous role must be specified.</td>
<td>N/A</td>
</tr>
<tr>
<td>InvalidArgument</td>
<td>Invalid parameter.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>InvalidBucketName</td>
<td>The specified bucket name in the request is invalid.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>InvalidBucket</td>
<td>The bucket to be accessed does not exist.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>InvalidBucketState</td>
<td>Invalid bucket status.</td>
<td>409 Conflict</td>
</tr>
<tr>
<td>InvalidBucketStoragePolicy</td>
<td>An invalid new policy is specified during bucket policy modification.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>InvalidDigest</td>
<td>The specified <strong>Content-MD5</strong> in the HTTP header is invalid.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>InvalidEncryptionAlgorithmError</td>
<td>Incorrect encryption algorithm.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>InvalidLocationConstraint</td>
<td>The location specified during bucket creation is invalid.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>InvalidPart</td>
<td>One or more specified parts are not found. The parts may not be uploaded or the specified entity tags (ETags) do not match the parts' ETags.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>InvalidPartOrder</td>
<td>Parts are not listed in ascending order by part number.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>InvalidPayer</td>
<td>All accesses to this object are disabled.</td>
<td>403 Forbidden</td>
</tr>
<tr>
<td>InvalidPolicyDocument</td>
<td>The content of the form does not meet the conditions specified in the policy document.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>InvalidRange</td>
<td>The requested range cannot be obtained.</td>
<td>416 Client Requested Range Not Satisfiable</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
<td>HTTP Status Code</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>InvalidRedirectLocation</td>
<td>Invalid redirect location.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>InvalidRequest</td>
<td>Invalid request.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>InvalidRequestBody</td>
<td>Invalid POST request body.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>InvalidSecurity</td>
<td>Invalid security credentials.</td>
<td>403 Forbidden</td>
</tr>
<tr>
<td>InvalidStorageClass</td>
<td>The specified storage class is invalid.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>InvalidTargetBucketForLogging</td>
<td>The delivery group has no ACL permission for the target bucket.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>InvalidURI</td>
<td>Cannot resolve the specified uniform resource identifier (URI).</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>KeyTooLong</td>
<td>The provided key is too long.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>MalformedACLLError</td>
<td>The provided XML has bad syntax or does not meet the format requirements.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>MalformedError</td>
<td>The XML format in the request is incorrect.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>MalformedLoggingStatus</td>
<td>The XML format of <strong>Logging</strong> is incorrect.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>MalformedPolicy</td>
<td>The bucket policy failed the check.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>MalformedPOSTRequest</td>
<td>The body of the POST request is in an incorrect format.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>MalformedQuotaError</td>
<td>The Quota XML format is incorrect.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>MalformedXML</td>
<td>This error code is returned after you send an XML file in incorrect format, stating &quot;The XML you provided was not well-formed or did not validate against our published schema.&quot;</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>MaxMessageLengthExceeded</td>
<td>The request is too long.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>MaxPostPreDataLengthExceededError</td>
<td>The POST request fields prior to the file to be uploaded are too large.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
<td>HTTP Status Code</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>MetadataTooLarge</td>
<td>The size of the metadata header has exceeded the upper limit.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>MethodNotAllowed</td>
<td>The specified method is not allowed against the requested resource.</td>
<td>405 Method Not Allowed</td>
</tr>
<tr>
<td>MissingContentLength</td>
<td>The HTTP header <code>Content-Length</code> is not provided.</td>
<td>411 Length Required</td>
</tr>
<tr>
<td>MissingRegion</td>
<td>No region in the request and no default region in the system.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>MissingRequestBodyError</td>
<td>This error code is returned after you send an empty XML file, stating &quot;Request body is empty.&quot;</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>MissingRequiredHeader</td>
<td>Required headers missing in the request.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>MissingSecurityHeader</td>
<td>A required header is not provided.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>NoSuchBucket</td>
<td>The specified bucket does not exist.</td>
<td>404 Not Found</td>
</tr>
<tr>
<td>NoSuchBucketPolicy</td>
<td>No bucket policy exists.</td>
<td>404 Not Found</td>
</tr>
<tr>
<td>NoSuchCORSConfiguration</td>
<td>No CORS configuration exists.</td>
<td>404 Not Found</td>
</tr>
<tr>
<td>NoSuchCustomDomain</td>
<td>The requested user domain does not exist.</td>
<td>404 Not Found</td>
</tr>
<tr>
<td>NoSuchKey</td>
<td>The specified key does not exist.</td>
<td>404 Not Found</td>
</tr>
<tr>
<td>NoSuchLifecycleConfiguration</td>
<td>The requested Lifecycle does not exist.</td>
<td>404 Not Found</td>
</tr>
<tr>
<td>NoSuchPolicy</td>
<td>The specified policy name does not exist.</td>
<td>404 Not Found</td>
</tr>
<tr>
<td>NoSuchUpload</td>
<td>The specified multipart upload does not exist. The upload ID does not exist or the multipart upload has been aborted or completed.</td>
<td>404 Not Found</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
<td>HTTP Status Code</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>NoSuchVersion</td>
<td>The specified version ID does not match any existing version.</td>
<td>404 Not Found</td>
</tr>
<tr>
<td>NoSuchWebsiteConfiguration</td>
<td>The requested website does not exist.</td>
<td>404 Not Found</td>
</tr>
<tr>
<td>NotImplemented</td>
<td>The provided header implies a function that is unavailable.</td>
<td>501 Not Implemented</td>
</tr>
<tr>
<td>NotSignedUp</td>
<td>Your account is not signed up for OBS. OBS is available only after you sign up.</td>
<td>403 Forbidden</td>
</tr>
<tr>
<td>OperationAborted</td>
<td>A conflicting operation is being performed on this resource. Retry later.</td>
<td>409 Conflict</td>
</tr>
<tr>
<td>PermanentRedirect</td>
<td>The requested bucket must be addressed using a specified endpoint. Send all future requests to the endpoint.</td>
<td>301 Moved Permanently</td>
</tr>
<tr>
<td>PreconditionFailed</td>
<td>At least one of the specified preconditions is not met.</td>
<td>412 Precondition Failed</td>
</tr>
<tr>
<td>Redirect</td>
<td>The request is temporarily redirected.</td>
<td>307 Moved Temporarily</td>
</tr>
<tr>
<td>RequestIsNotMultiPartContent</td>
<td>A bucket POST request must contain an enclosure-type multipart or the form-data.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>RequestTimeout</td>
<td>The socket connection to the server has no reads or writes within the timeout period.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>RequestTimeTooSkewed</td>
<td>The request time and the server's time differ a lot.</td>
<td>403 Forbidden</td>
</tr>
<tr>
<td>RequestTorrentOfBucketError</td>
<td>Requesting the bucket's torrent file is not allowed.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>ServiceNotImplemented</td>
<td>The request method is not implemented by the server.</td>
<td>501 Not Implemented</td>
</tr>
<tr>
<td>ServiceNotSupported</td>
<td>The request method is not supported by the server.</td>
<td>409 Conflict</td>
</tr>
<tr>
<td>ServiceUnavailable</td>
<td>The server is overloaded or has internal errors.</td>
<td>503 Service Unavailable</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
<td>HTTP Status Code</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>SignatureDoesNotMatch</td>
<td>The provided signature does not match the signature calculated by OBS. Check your AK and SK and signature calculation method.</td>
<td>403 Forbidden</td>
</tr>
<tr>
<td>SlowDown</td>
<td>Too frequent requests. Reduce your request frequency.</td>
<td>503 Service Unavailable</td>
</tr>
<tr>
<td>System Capacity Not enough</td>
<td>Insufficient system space.</td>
<td>403 Forbidden</td>
</tr>
<tr>
<td>TooManyCustomDomains</td>
<td>Too many user domains are configured.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>TemporaryRedirect</td>
<td>The request is redirected to the bucket while the domain name server (DNS) is being updated.</td>
<td>307 Moved Temporarily</td>
</tr>
<tr>
<td>TooManyBuckets</td>
<td>You have attempted to create more buckets than allowed.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>TooManyObjectCopied</td>
<td>The number of copied users' objects exceeds the upper limit.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>TooManyWrongSignature</td>
<td>The request is rejected due to high-frequency errors.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>UnexpectedContent</td>
<td>This request does not support content.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>UnresolvableGrantByEmailAddress</td>
<td>The provided email address does not match any recorded accounts.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>UserKeyMustBeSpecified</td>
<td>The user's AK is not carried in the request.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>WebsiteRedirect</td>
<td>The website request lacks bucketName.</td>
<td>301 Moved Permanently</td>
</tr>
<tr>
<td>KMS.DisabledException</td>
<td>The master key is disabled in SSE-KMS mode.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>KMS.NotFoundException</td>
<td>The master key does not exist in SSE-KMS mode.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>RestoreAlreadyInProgress</td>
<td>The archive objects are being restored. The request conflicts with another one.</td>
<td>409 Conflict</td>
</tr>
<tr>
<td>ObjectHasAlreadyRestored</td>
<td>The objects have been restored and the retention period of the objects cannot be shortened.</td>
<td>409 Conflict</td>
</tr>
</tbody>
</table>
### Error Code

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
<th>HTTP Status Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvalidObjectState</td>
<td>The restored object is not an Archive object.</td>
<td>403 Forbidden</td>
</tr>
<tr>
<td>InvalidTagError</td>
<td>An invalid tag is provided when configuring the bucket tag.</td>
<td>400 Bad Request</td>
</tr>
<tr>
<td>NoSuchTagSet</td>
<td>The specified bucket is not configured with a tag.</td>
<td>404 Not Found</td>
</tr>
</tbody>
</table>

### 17.2 SDK Error Handling

SDK errors include: errors returned by SDK functions and parameters, as well as errors 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

### 17.3 Log Analysis

#### Log Path

The OBS C SDK log path is specified by the LogPath field in the `OBS.ini`. By default, the log path is stored in the `logs` directory at the same level as the `lib` directory of the C SDK dynamic library. To locate a fault, you only need to view the run log `eSDK-OBS-API-*.*-C.run.log` in the `logs` directory of the same level.

#### Log Format

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

<table>
<thead>
<tr>
<th>Run logs</th>
</tr>
</thead>
</table>

#### 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
;Product Name
sdkname=eSDK-OBS-API-Linux-C
[LogPath]
;Log Path is relative to the path of configuration file
LogPath=../logs
```

17.4 Unmatched Signatures

If the HTTP status code obtained from the request response is 403 and the OBS server-side error code is OBS_STATUS_SignatureDoesNotMatch, check whether the AK/SK are correct.
## Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>What's New</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-03-30</td>
<td>This is the third official release. In <strong>2.2 Setting Up an OBS Environment</strong>, added the content related to the temporary access keys.</td>
</tr>
<tr>
<td>2018-08-31</td>
<td>This is the second official release. Reconstructed APIs.</td>
</tr>
<tr>
<td>2018-06-04</td>
<td>This is the first official release.</td>
</tr>
</tbody>
</table>