Object Storage Service

SDK Reference

SDK Reference

Java-SDK

Preface

Introduction

This document introduces the installation and use of the OSS Java SDK (JAVA SDK 2.0.5 particularly). This document assumes that you have already subscribed to the AliCloud OSS service and created an Access Key ID and Access Key Secret. In the document, ID represents the Access Key ID and KEY indicates the Access Key Secret. If you have not yet subscribed to or do not know about the OSS service, please log into the OSS Product Homepage for more help.

SDK Download

- Java SDK 2.4.0：aliyun_java_sdk_20161121.zip
- GitHub：https://github.com/aliyun/aliyun-oss-java-sdk

Version Revisions

Java SDK (2015-07-10) Version 2.0.5

Updates:

1. Added Append Object support.
2. Added HTTPS support.
3. Added the encoding-type parameter in the DeleteObject and ListObjects interfaces for users to specify the object name encoding method.
4. Removed the mandatory check of the Expires response header date format, fixing the issue where the system could not parse Expires response headers without using the GMT date format.

Java SDK (2015-05-29) Version 2.0.4

Updates:

- Added STS support.
- Added a test Demo Jar; for specific usage, refer to Readme.txt in the demo folder.
- Modified the CreateBucket logic to allow users to specify the Bucket ACL during creation.
- Modified the bucket name inspection rules, allow for operations on existing buckets with underlines, but not the creation of buckets with underlines.
- Optimized the HTTP connection pool, enhancing its concurrent processing capability.

Java SDK (2015-04-24) Version 2.0.3

Updates:

- Added the deleteObjects interface to batch delete object lists.
- Added the doesObjectExist interface to check for the existence of objects under the specified bucket.
- Added the switchCredentials interface, allowing users to switch the credentials of existing OSSClient instances.
- Added the OSSClient constructor with CredentialsProvider to allow for more customization of the CredentialsProvider.
- Fixed the bug that occurred when the Source Key in the copyObject interface contained the plus sign special character.
- Adjusted the OSSException/ClientException information display format.

Java SDK (2015-03-23) Version 2.0.2

Updates:

- Added the GeneratePostPolicy/calculatePostSignature interface, used to generate the Policy string and Post signature for Post requests.
- Support for Bucket Lifecycle.
- Added the URL signature-based Put/GetObject overload interface.
- Support for PutObject, UploadPart, and Chunked encoding data upload.
- Fixed several bugs.

Java SDK (2015-01-15) Version 2.0.1

Updates:

- Support for Cname, allowing users to specify which domain names are retained
- Support for user ContentType and ContentMD5 specification during URI generation.
- Addressed the problem where CopyObject requests did not support server-side encryption.

- Changed the UserAgent format.
- Expanded the Location constant.
- Added some samples.

Java SDK Development Kit (2014-11-13) Version 2.0.0

Updates:

- Upload Part Copy function.
- OSS Java SDK source code.
- Some sample codes.

This version is dependent on the version prior to modification. Using the previous SDK program version with this Java SDK requires you to change the package name reference. The package names com.aliyun.openservices.* and com.aliyun.openservices.oss.* should be changed to com.aliyun.oss.*. For example:Using the previous Java SDK version

```
import com.aliyun.openservices.ClientConfiguration;
import com.aliyun.openservices.ClientException;
import com.aliyun.openservices.ServiceException;
import com.aliyun.openservices.oss.OSSClient;
import com.aliyun.openservices.oss.OSSErrorCode;
import com.aliyun.openservices.oss.OSSException;
```

Using the 2.0.0 Java SDK version

```
import com.aliyun.oss.ClientConfiguration;
import com.aliyun.oss.ClientException;
import com.aliyun.oss.ServiceException;
import com.aliyun.oss.OSSClient;
import com.aliyun.oss.OSSErrorCode;
import com.aliyun.oss.OSSException;
```

Installation

Directly Using the JAR Package in Eclipse

Steps:

- Download Open Service Java SDK from the official website.
- Decompress the file.
- Copy the file aliyun-sdk-oss-<versionId>.jar from the post-decompression folder and all files in the lib folder to your project folder.

- In Eclipse, right-click on Project -> Properties -> Java Build Path -> Add JARs.
- Select all the JAR files you copied.
- After completing the steps above, you can use the OSS JAVA SDK in the project.

Using SDKs in Maven Projects

It is very easy to use the JAVA SDK in Maven projects. You just have to add the dependencies in the pom.xml file. Using version 2.0.2 as an example, enter the following content in the dependencies tab:

```
<dependency>
    <groupId>com.aliyun.oss</groupId>
    <artifactId>aliyun-sdk-oss</artifactId>
    <version>2.0.2</version>
</dependency>
```

Quick Start

In this chapter, you will learn how to use the basic functions of the OSS Java SDK.

Step-1. Initialize an OSSClient

SDK OSS operations are performed through OSSClients. The code below creates an OSSClient object:

```
import com.aliyun.oss.OSSClient;

public class Sample {

    public static void main(String[] args) {
        String accessKeyId = "<key>";
        String accessKeySecret = "<secret>";
        // Uses Hangzhou as an example
        String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";

        // Initializes an OSSClient
        OSSClient client = new OSSClient(endpoint,accessKeyId, accessKeySecret);

        // The following is the calling code...
        ...
    }
}
```

In the above code, the variables accessKeyId and accessKeySecret are allocated to the user by the system. They are called the ID pair and used to identify the user. They may be used to perform signature verification when your AliCloud account or RAM account accesses OSS. For more

information on the OSSClient, refer to OSSClient.

Step-2. Creating Buckets

Buckets are the OSS global namespace. They are equivalent to a data container and can store numerous objects. You can create a bucket with the following code:

```
public void createBucket(String bucketName) {

    // Initializes an OSSClient
    OSSClient client = new OSSClient(endpoint,accessKeyId, accessKeySecret);

    // Creates a Bucket
    client.createBucket(bucketName);
}
```

For bucket naming rules, refer to the naming rules in Bucket.

Step-3. Upload objects

Objects are the basic data elements in OSS. You can simply think of them as files. The code below will upload an object:

```
public void putObject(String bucketName, String key, String filePath) throws FileNotFoundException {

    // Initializes an OSSClient
    OSSClient client = new OSSClient(endpoint,accessKeyId, accessKeySecret);

    // Retrieves specified file input stream
    File file = new File(filePath);
    InputStream content = new FileInputStream(file);

    // Creates Metadata for the object to upload
    ObjectMetadata meta = new ObjectMetadata();

    // You must set the ContentLength
    meta.setContentLength(file.length());

    // Uploads the object
    PutObjectResult result = client.putObject(bucketName, key, content, meta);

    // Prints the ETag
    System.out.println(result.getETag());
}
```

The object is uploaded to OSS through InputStream form. In the above example, we can see that, each time you upload an object, you must specify the ObjectMetadata associated with it. ObjectMetaData is the user's description of the object, composed of a series of name-value pairs.

Here, ContentLength is a required setting, used to allow the SDK to correctly identify the size of the object to be uploaded. In order to ensure the consistency of files uploaded to the server with the version on the local disk, users can set ContentMD5. OSS will calculate the MD5 value for the upload data and compare it with the MD5 value uploaded by the user. If they are inconsistent, the system will return the InvalidDigest error code. For object naming rules, refer to the naming rules in Object. For more information about uploading objects, refer to uploading objects in Object.

Step-4. Listing all objects

When you complete a series of uploads, you may need to view which objects are in a bucket. This can be done with the following program:

```
public void listObjects(String bucketName) {

    // Initializes an OSSClient
    OSSClient client = new OSSClient(endpoint,accessKeyId, accessKeySecret);

    // Retrieves information for all objects in the specified bucket
    ObjectListing listing = client.listObjects(bucketName);

    // Traverses all objects
    for (OSSObjectSummary objectSummary : listing.getObjectSummaries()) {
        System.out.println(objectSummary.getKey());
    }
}
```

The listObjects method returns the ObjectListing object. This contains the returned results for this listObject request. Here, we can use the getObjectSummaries method in ObjectListing to retrieve all object description information.

Step-5. Retrieving a specified object

You can refer to the code below to easily retrieve an object:

```
public void getObject(String bucketName, String key) throws IOException {

    // Initializes an OSSClient
    OSSClient client = new OSSClient(endpoint,accessKeyId, accessKeySecret);
    // Retrieves an object; the returned result is an OSSObject object
    OSSObject object = client.getObject(bucketName, key);
    // Retrieves an object input stream
    InputStream objectContent = object.getObjectContent();
    // Processes the object
    ...
    // Closes the stream
    objectContent.close();
}
```

When you call the OSSClient's getObject method, it returns an OSSObject object, containing various object information. Using the OSSObject's getObjectContent method, you can retrieve the returned object input stream and obtain the object content by reading it. When you have finished the operation, close the stream.

OSSClient

The OSSClient is the Java client of OSS services. It provides a series of functions to be called for interaction with OSS services.

Creating the OSSClient

It is very easy to create an OSSClient, as shown in the code below:

```
String key = "<key>";
String secret = "<secret>";
// Uses Hangzhou as an example
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";

OSSClient client = new OSSClient(endpoint, accessKeyId, accessKeySecret);
```

It is critical that the user import the AccessKey and access the bucket endpoint. If the user wishes to use HTTPS protocol, the endpoint must start with https://.

Configuring the OSSClient

To configure detailed parameters for the OSSClient, you can import a ClientConfiguration object when creating the OSSClient. ClientConfiguration is the OSS service configuration type. It allows you to configure a proxy for the client, the maximum number of connections, and other parameters.

Using a Proxy

The code below allows the client to use a proxy to access the OSS service:

```
// Creates a ClientConfiguration instance
ClientConfiguration conf = new ClientConfiguration();

// Configures the proxy for the local port 8080
conf.setProxyHost("127.0.0.1");
conf.setProxyPort(8080);

// Creates the OSSClient
client = new OSSClient(endpoint, accessKeySecret, accessKeySecret, conf);
```

The code above enables all operations on the client to be executed by proxy through port 8080 at 127.0.0.1. For proxies that authenticate users, you can configure the usernames and passwords:

```
// Creates a ClientConfiguration instance
ClientConfiguration conf = new ClientConfiguration();

// Configures the proxy for the local port 8080
conf.setProxyHost("127.0.0.1");
conf.setProxyPort(8080);

//Sets the username and password
conf.setProxyUsername("username");
conf.setProxyPassword("password");
```

Setting Network Parameters

We can use ClientConfiguration to set some network parameters:

```
ClientConfiguration conf = new ClientConfiguration();

// Sets the maximum number of HTTP connections to 10
conf.setMaxConnections(10);

// Sets TCP connection timeout to 5000 milliseconds
conf.setConnectionTimeout(5000);

// Sets the maximum number of retries to 3
conf.setMaxErrorRetry(3);

// Sets the Socket data transmission time out to 2,000 milliseconds
conf.setSocketTimeout(2000);
```

Using ClientConfiguration we can set the following parameters:

Parameter	Description
UserAgent	The user proxy, specifies the HTTP User-Agent header. The default value is "aliyun-sdk-java"
ProxyHost	The proxy's host address
ProxyPort	The proxy's port
ProxyUsername	The username verified by the proxy
ProxyPassword	The password verified by the proxy
ProxyDomain	The Windows domain name to access the NTLM verified proxy
ProxyWorkstation	The name of the Windows workstation for the NTLM proxy
MaxConnections	The maximum number of HTTP connections that can be enabled. The default value is 1024

SocketTimeout	The time out of enabling connections for data transmission (unit: milliseconds). The default value is 50000 milliseconds
ConnectionTimeout	The connection establishment timeout time (unit: milliseconds).The default value is 50000 milliseconds
MaxErrorRetry	The maximum number of times failed requests can be retried. The default value is 3

Bucket

OSS uses buckets as the namespaces of user files and also as the management objects for advanced functions such as charging, permission control, and log recording. The bucket name must be globally unique in the entire OSS and cannot be changed. Every object stored on the OSS must be included in a bucket. One application, such as an image sharing website, can correspond to one or more buckets. A user can create a maximum of 10 buckets, but there is no limit on the number of objects in each bucket. Each bucket can store up to 2 PB of data.

Naming Rules

The bucket naming rules are as follows:

- It can only contain lower-case letters, numbers, and dashes (-).
- It must start with a lower-case letter or number.
- The length must be 3-63 bytes

Creating Buckets

We can use the code below to create a bucket:

```
String bucketName = "my-bucket-name";

// Initializes an OSSClient
OSSClient client = ...;

// Creates a Bucket
client.createBucket(bucketName);
```

Because bucket names are globally unique, do your best to ensure your bucket names are not the same as other people's.

Listing all Buckets of a User

The code below will list all the buckets of the user:

```
// Retrieves the user's bucket list
List<Bucket> buckets = client.listBuckets();

// Traverses buckets
for (Bucket bucket : buckets) {
    System.out.println(bucket.getName());
}
```

CNAME Access

After a user directs his own domain name's CNAME to the domain name of one of his buckets, he can access OSS through his domain name:

```
// For example, your domain name is "http://cname.com" and you direct the CNAME to your bucket domain name
"mybucket.oss-cn-hangzhou.aliyuncs.com"
OSSClient client = new OSSClient("http://cname.com/", /* accessKeyId */, /* accessKeySecret */);

PutObjectResult result = client.putObject("mybucket", /* key */, /* input */, /* metadata */);
```

Users just need to change the endpoint originally expected to be entered in the bucket to the post-CNAME domain name when creating an OSSClient instance. At the same time, users must note that, when using this OSSClient instance for subsequent operations, the bucket name can only be filled by the indicated bucket name.

> If a VPC user wishes to access OSS through a domain name that does not end with aliyuncs.com, he can use the setCnameExcludeList in the ClientConfiguration to set an endpoint and avoid using the CNAME method to access OSS.

Determining If a Bucket Exists

To determine if a bucket exists, we can use the following code:

```
String bucketName = "your-bucket-name";

// Gets information on existing buckets
boolean exists = client.doesBucketExist(bucketName);

// Outputs results
if (exists) {
    System.out.println("Bucket exists");
} else {
```

```
    System.out.println("Bucket not exists");
}
```

Setting Bucket ACL

To set the bucket ACL, we can use the following code:

```
String bucketName = "your-bucket-name";

//Uses a bucket with private permission as an example
client.setBucketAcl(bucketName,CannedAccessControlList.Private);
```

Retrieving Bucket ACL

To retrieve the bucket ACL, we can use the following code:

```
String bucketName = "your-bucket-name";
AccessControlList accessControlList = client.getBucketAcl(bucketName);

//You can print the results or confirm them on the console
System.out.println(accessControlList.toString());
```

Retrieving Bucket Addresses

To retrieve the bucket address, we can use the following code:

```
String bucketName = "your-bucket-name";

// Retrieves the bucket address
String location = client.getBucketLocation(bucketName);
System.out.println(location);
```

Deleting Buckets

The following code deletes a bucket:

```
String bucketName = "your-bucket-name";

// Deletes the bucket
client.deleteBucket(bucketName)
```

You must note that, if the bucket is not empty (the bucket contains objects), it cannot be deleted. You must delete all objects in a bucket before deleting the bucket.

Object

In OSS, objects are the basic data units for user operation. The maximum size of a single object may vary depending on the data uploading mode. The size of an object cannot exceed 5 GB in the Put Object mode or 48.8 TB in the multipart upload mode. An object includes the key, meta, and data. The key is the object name; meta is the user's description of the object, composed of a series of name-value pairs; and data is the object data.

Naming Rules

Object naming rules:

- It uses UTF-8 encoding
- The length must be 1-1023 bytes
- It cannot start with "/" or "\"
- It cannot contain "\r" or "\n" line breaks

Uploading Objects

Simplest Upload

The code is as follows:

```
public void putObject(String bucketName, String key, String filePath) throws FileNotFoundException {
    // Initializes an OSSClient
    OSSClient client = ...;
    // Retrieves specified file input stream
    File file = new File(filePath);
    InputStream content = new FileInputStream(file);
    // Creates Metadata for the object to upload
    ObjectMetadata meta = new ObjectMetadata();
    // You must set the ContentLength
    meta.setContentLength(file.length());

    // Uploads the object
    PutObjectResult result = client.putObject(bucketName, key, content, meta);

    // Prints the ETag
    System.out.println(result.getETag());
}
```

The object is uploaded to OSS in InputStream form. In the above example, we can see that, each time you upload an object, you must specify the ObjectMetadata associated with the object.

ObjectMetaData is the user's description of the object, composed of a series of name-value pairs. Here, ContentLength is required for the SDK to correctly identify the size of the object to be uploaded. In order to ensure the consistency between files uploaded to the server and local files, users can set up ContentMD5. OSS will calculate and compare the MD5 value for the upload data with the MD5 value uploaded by the user. If they are inconsistent, the system will return the InvalidDigest error code.

Creating Simulated Folders

The OSS service does not use folders. All elements are stored as objects. However, users can create simulated folders using the following code:

```
String bucketName = "your-bucket-name";
//The name of the folder to be created, which must satisfy the object naming rules and end with "/"
String objectName = "folder_name/";
OSSClient client = new OSSClient(OSS_ENDPOINT, ACCESS_ID, ACCESS_KEY);
ObjectMetadata objectMeta = new ObjectMetadata();
/*Here, the size is 0. Note that OSS does not use folders. Instead, you can simulate a folder using an object with a size of 0, but the dataStream may still have data
 */
byte[] buffer = new byte[0];
ByteArrayInputStream in = new ByteArrayInputStream(buffer);
objectMeta.setContentLength(0);
try {
    client.putObject(bucketName, objectName, in, objectMeta);
} finally {
    in.close();
}
```

To create simulated folders in the OSS Console, you can create an object with a size of 0. This object can still be uploaded and downloaded. The console will display any object ending with "/" as a folder. Therefore, users can create simulated folders this way. For accessing folders, refer to the folder simulation function

Setting the Object's Http Header

The OSS service allows users to customize the object Http Header. The following code sets the expiration time for the object:

```
// Initializes an OSSClient
OSSClient client = ...;

// Initializes the upload input stream
InputStream content = ...;

// Creates Metadata for the object to upload
ObjectMetadata meta = new ObjectMetadata();
```

```
// Sets the ContentLength to 1000
meta.setContentLength(1000);

// Sets the object to expire after 1 hour
Date expire = new Date(new Date().getTime() + 3600 * 1000);
meta.setExpirationTime(expire);
client.putObject(bucketName, key, content, meta);
```

The Java SDK supports four types of Http Headers: Cache-Control, Content-Disposition, Content-Encoding, and Expires. For details on the headers, please see RFC2616.

User-Defined Metadata

The OSS allows users to define metadata to describe the object. For example:

```
// Sets the custom metadata name value as my-data
meta.addUserMetadata("name", "my-data");

// Uploads the object
client.putObject(bucketName, key, content, meta);
```

In the above code, the user has defined a metadata with its name as "name" and its value as "my-data". When downloading this object, users will also obtain the metadata. A single object can have multiple similar parameters, but the total size of all user meta cannot exceed 2 KB.

NOTE: The user meta name is not case sensitive. For instance, when a user uploads an object and defines the metadata name as "Name", the parameter stored in the header will be: "x-oss-meta-name". Therefore, when accessing the object, just use parameters named "name". However, if the stored parameter is "name", no information can be found for the parameter and the system will return "Null"

Upload Through Chunked Encoding

When the length of the content to be uploaded is unknown (e.g. SocketStream is being received and uploaded at the same time as the data source for uploading until the Socket ends), chunked encoding applies.

Chunked adopts the following encoding method:

```
Chunked-Body = *chunk
"0" CRLF
footer
CRLF
chunk = chunk-size [chunk-ext] CRLF
chunk-data CRLF

hex-no-zero = <HEX excluding "0">

chunk-size = hex-no-zero * HEX
```

```
chunk-ext = *( ";" chunk-ext-name [ "=" chunk-ext-value ] )
chunk-ext-name = token
chunk-ext-val = token | quoted-string
chunk-data = chunk-size(OCTET)

footer = *entity-header
```

The code contains several chunks, ending with a chunk marked with a length of 0. Each chunk is composed of two parts. The first part indicates the length of the chunk and the unit of the length (generally not specified). The second part contains the content of the indicated length. Each part is separated by CRLF. The last chunk with a length of 0 contains something called footer, which is a blank header.

putobject chunked encoding When explicitly setting the ContentLength attribute in the ObjectMetadata instance, the normal upload method applies (with the request body length determined by the Content-Length request header). Otherwise, the chunked encoding method applies.

```
OSSClient client = new OSSClient(endpoint, accessId, accessKey);

FileInputStream fin = new FileInputStream(new File(filePath));
// If content-length is not specified, chunked encoding is the default value.
PutObjectResult result = client.putObject(bucketName, key, fin, new ObjectMetadata());
```

Append Object

OSS allows users to directly append content to the end of an object using the Append Object method. This requires the object type to be Appendable. Use the Append Object operation when the created Object type is Appendable Object and the Put Object upload method for Normal Objects.

```
// Creates an OSSClient instance
OSSClient client = new OSSClient(ENDPOINT, ACCESS_ID, ACCESS_KEY);

// Initiates the first Append Object request; note that the first append operation requires the append location to be set to 0
final String fileToAppend = "<file to append at first time>";
AppendObjectRequest appendObjectRequest = new AppendObjectRequest(bucketName, key, new File(fileToAppend));

// Sets content-type; note that you can only set object meta for objects created using Append
ObjectMetadata meta = new ObjectMetadata();
meta.setContentType("image/jpeg");
appendObjectRequest.setMetadata(meta);

// Sets the append location to 0 and sends the Append Object request
appendObjectRequest.setPosition(0L);
AppendObjectResult appendObjectResult = client.appendObject(appendObjectRequest);

// Initiates the second Append Object request. The append location is set to the object length after the first append
final String fileToAppend2 = "<file to append at second time>";
```

```
appendObjectRequest = new AppendObjectRequest(bucketName, key, new File(fileToAppend2));

// Sets the append location as the size of the previously appended file and sends the Append object request
appendObjectRequest.setPosition(appendObjectResult.getNextPosition());
appendObjectResult = client.appendObject(appendObjectRequest);

OSSObject o = client.getObject(bucketName, key);
// The current size of this object is the total size of the two previously appended files
System.out.println(o.getObjectMetadata().getContentLength());
// The next append location is the total size of the two previously appended files
System.err.println(appendObjectResult.getNextPosition().longValue());
```

When using the Append upload method, the user must set the Position parameter correctly. When the user creates an Appendable Object, the append location is 0. When appending content to an Appendable Object, the append location is the object's current length. There are two methods to get the object length: One is to obtain it from the returned content following the append upload, as shown in the code above. The other is to use the getObjectMetadata operation described below to get the object's current length.

> Object meta settings only apply when Append Object is used to create an object. Subsequently, the Object meta can be changed by using the copy object interface described below (with the source and destination set as the same object).

Multipart Upload

OSS allows users to split an object into several requests for uploading to the server. Concerning the block upload content, refer to the Multipart Upload section in MultipartUpload.

List Bucket Objects

Listing Objects

```
public void listObjects(String bucketName) {

    // Initializes an OSSClient
    OSSClient client = ...;

    // Retrieves information for all objects in the specified bucket
    ObjectListing listing = client.listObjects(bucketName);

    // Traverses all objects
    for (OSSObjectSummary objectSummary : listing.getObjectSummaries()) {
        System.out.println(objectSummary.getKey());
    }
}
```

The listObjects method returns the ObjectListing object, which contains the returned results for this listObject request. Here, we can use the getObjectSummaries method in ObjectListing to retrieve all object description information (List<OSSObjectSummary>).

> NOTE: By default, if a bucket contains more than 100 objects, the first 100 will be returned and the IsTruncated parameter in the returned results will be true. The returned NextMarker can be used as the start point for next data access. The number of object entries returned can be increased by modifying the MaxKeys parameter or using the Marker parameter for separate access.

Extended Parameters

Generally, the ListObjectsRequest parameter provides more powerful functions. For example:

```
// Constructs the ListObjectsRequest request
ListObjectsRequest listObjectsRequest = new ListObjectsRequest(bucketName);

// Sets parameters
listObjectsRequest.setDelimiter("/");
listObjectsRequest.setMarker("123");
...

ObjectListing listing = client.listObjects(listObjectsRequest);
```

In the above code, we called the listObjects overload method and used the incoming ListObjectsRequest to complete the request. Setting the parameters in ListObjectsRequest creates many extended functions. The table below lists the names and actions of ListObjectsRequest parameters:

Name	Function
Delimiter	Used to group object name characters. All objects whose names contain the specified prefix and that appear between the Delimiter characters for the first time are used as a group of elements: CommonPrefixes.
Marker	Sets up the returned results to begin from the first entry after the Marker in alphabetical order.
MaxKeys	Limits the maximum number of objects returned for one request. If not specified, the default value is 100. The MaxKeys value cannot exceed 1000.
Prefix	requires the returned object key to be prefixed with prefix. Note that the keys returned from queries using a prefix will still contain the prefix.

Multiple iterations must be performed to traverse a whole batch of over 1,000 objects. During each iteration, the final object key of the last iteration can be used as the Marker in the current iteration.

Folder Function Simulation

We can use a combination of Delimiter and Prefix to simulate folder functions. Combinations of Delimiter and Prefix serve the following purposes: Setting Prefix as the name of a folder enumerates the files starting with this prefix, recursively returning all files and subfolders in this folder. When the Delimiter is set as "/", the returned values will enumerate the files in the folder and the subfolders will be returned in the CommonPrefixes section. Recursive files and folders in subfolders will not be displayed. If the bucket contains 4 files: oss.jpg, fun/test.jpg, fun/movie/001.avi, and fun/movie/007.avi. We use the "/" symbol as the separator for folders.

List All Bucket Files

To retrieve all files in a bucket, write the following:

```
// Constructs the ListObjectsRequest request
ListObjectsRequest listObjectsRequest = new ListObjectsRequest(bucketName);

// List Objects
ObjectListing listing = client.listObjects(listObjectsRequest);

// Traverses all objects
System.out.println("Objects:");
for (OSSObjectSummary objectSummary : listing.getObjectSummaries()) {
    System.out.println(objectSummary.getKey());
}

// Traverse all CommonPrefix
System.out.println("CommonPrefixs:");
for (String commonPrefix : listing.getCommonPrefixes()) {
    System.out.println(commonPrefix);
}
```

Output:

```
Objects:
fun/movie/001.avi
fun/movie/007.avi
fun/test.jpg
oss.jpg

CommonPrefixs:
```

Recursively List All Files in a Directory

We can set the Prefix parameter to retrieve all the files under a directory:

```
// Constructs the ListObjectsRequest request
ListObjectsRequest listObjectsRequest = new ListObjectsRequest(bucketName);

// Recursively lists all files in the fun directory
listObjectsRequest.setPrefix("fun/");

ObjectListing listing = client.listObjects(listObjectsRequest);

// Traverses all objects
System.out.println("Objects:");
for (OSSObjectSummary objectSummary : listing.getObjectSummaries()) {
    System.out.println(objectSummary.getKey());
}

// Traverse all CommonPrefix
System.out.println("\nCommonPrefixs:");
for (String commonPrefix : listing.getCommonPrefixes()) {
    System.out.println(commonPrefix);
}
```

Output:

```
Objects:
fun/movie/001.avi
fun/movie/007.avi
fun/test.jpg

CommonPrefixs:
```

List Files and Subdirectories in a Directory

Using Prefix and Delimiter together, we can list the files and subdirectories under a directory:

```
// Constructs the ListObjectsRequest request
ListObjectsRequest listObjectsRequest = new ListObjectsRequest(bucketName);

// "/" is the folder separator
listObjectsRequest.setDelimiter("/");

// Lists all files and folders in the fun directory
listObjectsRequest.setPrefix("fun/");

ObjectListing listing = client.listObjects(listObjectsRequest);

// Traverses all objects
System.out.println("Objects:");
for (OSSObjectSummary objectSummary : listing.getObjectSummaries()) {
    System.out.println(objectSummary.getKey());
}

// Traverse all CommonPrefix
```

```
System.out.println("\nCommonPrefixs:");
for (String commonPrefix : listing.getCommonPrefixes()) {
    System.out.println(commonPrefix);
}
```

Output:

```
Objects:
fun/test.jpg

CommonPrefixs:
fun/movie/
```

In the returned results, the ObjectSummaries list contains the files in the fun directory. The CommonPrefixs list shows all subfolders in the fun directory. Obviously, the files fun/movie/001.avi and fun/movie/007.avi are not listed, because they are in the movie directory under the fun folder.

Retrieving Objects

Simply Getting Object

The following code can be used to get and input an object into a stream:

```
public void getObject(String bucketName, String key) throws IOException {

    // Initializes an OSSClient
    OSSClient client = ...;

    // Retrieves an object; the returned result is an OSSObject object
    OSSObject object = client.getObject(bucketName, key);

    // Retrieves ObjectMeta
    ObjectMetadata meta = object.getObjectMetadata();

    // Retrieves an object input stream
    InputStream objectContent = object.getObjectContent();

    // Processes the object
    ...

    // Closes the stream
    objectContent.close();
}
```

OSSObject contains various object information, including the object's bucket, object name, metadata, and an input stream. The input stream can be used to get and store the object content into an object or the memory. ObjectMetadata contains the ETag, Http Header, and custom metadata defined when the object was uploaded.

Using GetObjectRequest to Retrieve Objects

For more functions, we can use GetObjectRequest to retrieve objects.

```
// Initializes an OSSClient
OSSClient client = ...;

// Creates GetObjectRequest
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, key);

// Retrieves 0-100 bytes of data
getObjectRequest.setRange(0, 100);

// Retrieves an object; the returned result is an OSSObject object
OSSObject object = client.getObject(getObjectRequest);
```

We can use the setRange method in getObjectRequest to return the object range. We can use this function for multipart download and resumable data transfer. GetObjectRequest can set the following parameters:

Parameter	Description
Range	Specifies the range of file transfer.
ModifiedSinceConstraint	If the specified time is earlier than the actual modification time, the file is transmitted normally. Otherwise, the system throws the 304 Not Modified exception.
UnmodifiedSinceConstraint	If the specified time is the same as or later than the actual modification time, the file is transmitted normally. Otherwise, the system throws the 412 precondition failed exception
MatchingETagConstraints	Enters an ETag group. If the entered expected ETag matches the object's ETag, the file is transmitted normally.Otherwise, the system throws the 412 precondition failed exception
NonmatchingEtagConstraints	Enters an ETag group. If the entered expected ETag does not match the object's ETag, the file is transmitted normally.Otherwise, the system throws the 304 Not Modified exception.
ResponseHeaderOverrides	Customizes some headers in the OSS return request.

ResponseHeaderOverrides provides a series of modifiable parameters which customize the headers returned by OSS. This is shown in the table below:

Parameter	Description
ContentType	OSS returns the requested content-type header

ContentLanguage	OSS returns the requested content-language header
Expires	OSS returns the requested expires header
CacheControl	OSS returns the requested cache-control header
ContentDisposition	OSS returns the requested content-disposition header
ContentEncoding	OSS returns the requested content-encoding header

Directly Downloading Objects to Files

We can use the code below to directly download objects to a specified file:

```
// Creates GetObjectRequest
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, key);

// Downloads the object to the file
ObjectMetadata objectMetadata = client.getObject(getObjectRequest, new File("/path/to/file"));
```

When the above method is used to download objects to files, the ObjectMetadata object is returned.

Only Retrieve ObjectMetadata

The getObjectMetadata method can be used only to get the ObjectMetadata, instead of the object entity. The code is as follows:

```
ObjectMetadata objectMetadata = client.getObjectMetadata(bucketName, key);
```

Deleting Objects

The following code deletes an object:

```
public void deleteObject(String bucketName, String key) {
    // Initializes an OSSClient
    OSSClient client = ...;

    // Deletes the object
    client.deleteObject(bucketName, key);
}
```

Copying Objects

Copying One Object

Using the copyObject method, we can copy a single object. The code is as follows:

```
public void copyObject(String srcBucketName, String srcKey, String destBucketName, String destKey) {
    // Initializes an OSSClient
    OSSClient client = ...;

    // Copies the object
    CopyObjectResult result = client.copyObject(srcBucketName, srcKey, destBucketName, destKey);

    // Prints the results
    System.out.println("ETag: " + result.getETag() + " LastModified: " + result.getLastModified());
}
```

The copyObject returns one CopyObjectResult object. This contains the ETag of the new object and the modification time. The objects copied with this method must be smaller than 1 GB. Otherwise the system will report an error. If the object is larger than 1 GB, use the Upload Part Copy method given below

Using CopyObjectRequest to Copy Objects

We can also use the CopyObjectRequest to copy objects:

```
// Initializes an OSSClient
OSSClient client = ...;

// Creates a CopyObjectRequest object
CopyObjectRequest copyObjectRequest = new CopyObjectRequest(srcBucketName, srcKey, destBucketName,
destKey);

// Sets new metadata
ObjectMetadata meta = new ObjectMetadata();
meta.setContentType("text/html");
copyObjectRequest.setNewObjectMetadata(meta);

// Copies the object
CopyObjectResult result = client.copyObject(copyObjectRequest);

System.out.println("ETag: " + result.getETag() + " LastModified: " + result.getLastModified());
```

CopyObjectRequest allows users to modify the destination object's ObjectMeta. The method provides ModifiedSinceConstraint, UnmodifiedSinceConstraint, MatchingETagConstraints, and NonmatchingEtagConstraints. These parameters are used similarly to those of GetObjectRequest. For details, refer to the parameters of GetObjectRequest.

NOTE: Copying data can modify the meta information of an existing object. If the copied source object address is the same as the destination object address, the source object's meta information will be replaced regardless of the x-oss -metadata -directive value

POST Method File Uploads

Back-end services are required to provide the Policy and Signature form fields for the front-end.

Generating POST Policy

The policy form field requested by POST is used to verify the validity of the request. As a JSON text segment encoded in UTF-8 and base64, policy indicates the conditions the POST request must satisfy. Although the post form field is optional for uploading public-read-write buckets, we strongly suggest using this field to limit POST requests. For details on the policy json string generation rules, see Post Policy in the API documentation Post Object.

Below is an example of policy string configuration:

```
{"expiration":"2015-02-25T14:25:46.000Z",
"conditions":[
    {"bucket":"oss-test2"},
    ["eq","$key","user/eric/${filename}"],
    ["starts-with","$key","user/eric"],
    ["starts-with","$x-oss-meta-tag","dummy_etag"],
    ["content-length-range",1,1024]
    ]
}
```

We can use the following code to generate the json string described above:

```
OSSClient client = new OSSClient(endpoint, accessId, accessKey);

Date expiration = DateUtil.parseIso8601Date("2015-02-25T14:25:46.000Z");
PolicyConditions policyConds = new PolicyConditions();
policyConds.addConditionItem("bucket", bucketName);
// The exact match condition “$” must be followed by braces
policyConds.addConditionItem(MatchMode.Exact, PolicyConditions.COND_KEY, "user/eric/\\${filename}");
// Adds a prefix match condition
policyConds.addConditionItem(MatchMode.StartWith, PolicyConditions.COND_KEY, "user/eric");
policyConds.addConditionItem(MatchMode.StartWith, "x-oss-meta-tag", "dummy_etag");
// Adds a range match condition
policyConds.addConditionItem(PolicyConditions.COND_CONTENT_LENGTH_RANGE, 1, 1024);

// Generates the Post Policy string
String postPolicy = client.generatePostPolicy(expiration, policyConds);
System.out.println(postPolicy);

// Calculates the policy's Base64 encoding
byte[] binaryData = postPolicy.getBytes("utf-8");
String encodedPolicy = BinaryUtil.toBase64String(binaryData);
System.out.println(encodedPolicy);
```

After generating the policy string, note that the policy in the POST form field must be based on

Base64 encoding.

Generating Post Signature

Moreover, for POST uploading, a Post Signature must be generated to verify the request's validity. Refer to the following code.

```
//Imports the original Post Policy json string to generate the postSignature
String postSignature = client.calculatePostSignature(postPolicy);
System.out.println(postSignature);
```

Multipart Upload

Besides using the putObject interface to upload files to OSS, the OSS also provides a Multipart Upload mode.You can apply the Multipart Upload mode in the following scenarios (but not limited to the following):

- Where breakpoint uploads are needed.
- Uploading an object larger than 100MB.
- In poor network conditions, when the connection with the OSS server is frequently broken.
- When, before uploading the file, you cannot determine its size.

Below, we will give a step-by-step introduction to Multipart Upload.

Step-By-Step Multipart Upload

Initializing Multipart Upload

We use the initiateMultipartUpload method to initialize a multipart upload task:

```
String bucketName = "your-bucket-name";
String key = "your-key";

// Initializes an OSSClient
OSSClient client = ...;
// Starts Multipart Upload
InitiateMultipartUploadRequest initiateMultipartUploadRequest = new
InitiateMultipartUploadRequest(bucketName, key);
InitiateMultipartUploadResult initiateMultipartUploadResult =
client.initiateMultipartUpload(initiateMultipartUploadRequest);
// Prints UploadId
System.out.println("UploadId: " + initiateMultipartUploadResult.getUploadId());
```

We use InitiateMultipartUploadRequest to specify the name and bucket of the object to upload. In InitiateMultipartUploadRequest, you can also set the ObjectMetadata, but are not required to specify the ContentLength. The initiateMultipartUpload returned result includes the UploadId. This is the unique identifier of a multipart upload task. We will use this in subsequent operations.

Next, we can use two methods for uploading parts: use Upload Part to upload from the local disk or use Upload Part copy to get a copy of an object from a bucket.

Upload Part Local Upload

Next, we will multipart upload the local file. Let us assume that there is one file in the path /path/to/file.zip. Because it is large, we want to multipart upload it to OSS.

```
// Sets each part to 5M
final int partSize = 1024 * 1024 * 5;
File partFile = new File("/path/to/file.zip");
// Calculates the number of parts
int partCount = (int) (partFile.length() / partSize);
if (partFile.length() % partSize != 0){
    partCount++;
}
// Creates a list to save the ETag and PartNumber of each part after it is uploaded
List<PartETag> partETags = new ArrayList<PartETag>();
for(int i = 0; i < partCount; i++){
    // Retrieves the file stream
    FileInputStream fis = new FileInputStream(partFile);
    // Skips to the start of each part
    long skipBytes = partSize * i;
    fis.skip(skipBytes);
    // Calculates the size of each part
    long size = partSize < partFile.length() - skipBytes ?
            partSize : partFile.length() - skipBytes;
    // Creates an UploadPartRequest and performs multipart upload
    UploadPartRequest uploadPartRequest = new UploadPartRequest();
    uploadPartRequest.setBucketName(bucketName);
    uploadPartRequest.setKey(key);
    uploadPartRequest.setUploadId(initiateMultipartUploadResult.getUploadId());
    uploadPartRequest.setInputStream(fis);
    uploadPartRequest.setPartSize(size);
    uploadPartRequest.setPartNumber(i + 1);
    UploadPartResult uploadPartResult = client.uploadPart(uploadPartRequest);
    // Saves the returned PartETags to the List.
    partETags.add(uploadPartResult.getPartETag());
    // Closes the file
    fis.close();
}
```

The main idea of this program is to call the uploadPart method to upload each part. However, you must note the following:

- In the uploadPart method, all parts except the last one must be larger than 100KB.However,

the Upload Part interface does not immediately verify the size of the uploaded part (because it does not know whether the part is the last one). It verifies the size of the uploaded part only when Multipart Upload is completed.
- OSS will put the MD5 value of the part data received by the server in the ETag header and return it to the user.
- In order to ensure that the data transmitted over the network is free from errors, users can set ContentMD5. OSS will calculate the MD5 value for the uploaded data and compare it with the MD5 value uploaded by the user. If they are inconsistent, the system will return the InvalidDigest error code.
- The part number range is 1~10000. If the part number exceeds this range, the OSS will return the InvalidArgument error code.
- When each part is uploaded, it will take the stream to the location corresponding to the start of the next part.
- After each part is uploaded, the OSS returned results will include the PartETag object. This is the combination of the ETag and PartNumber of the uploaded part. This will be used in subsequent steps, so we need to save it. Generally, we will save these PartETag objects in the List.

Upload Part Local Chunked Upload

Chunked encoding is also supported for multipart uploads

```
File file = new File(filePath);
// Sets each part to 5M
final int partSize = 5 * 1024 * 1024;
int fileSize = (int) file.length();
// Calculates the number of parts
final int partCount = (file.length() % partSize != 0) ? (fileSize / partSize + 1) : (fileSize / partSize);
List<PartETag> partETags = new ArrayList<PartETag>();

for (int i = 0; i < partCount; i++) {
    InputStream fin = new BufferedInputStream(new FileInputStream(file));
    fin.skip(i * partSize);
    int size = (i + 1 == partCount) ? (fileSize - i * partSize) : partSize;

    UploadPartRequest req = new UploadPartRequest();
    req.setBucketName(bucketName);
    req.setKey(key);
    req.setPartNumber(i + 1);
    req.setPartSize(size);
    req.setUploadId(uploadId);
    req.setInputStream(fin);
    req.setUseChunkEncoding(true);  // Uses chunked encoding

    UploadPartResult result = client.uploadPart(req);
    partETags.add(result.getPartETag());

    fin.close();
}
```

In UploadPartRequest instances, we can set setUseChunkEncoding(true) to upload using chunked encoding.

Upload Part Copy

Using Upload Part Copy, we copy data from an existing object to upload an object. When copying an object larger than 500MB, we suggest using the Upload Part Copy method.

```
ObjectMetadata objectMetadata = client.getObjectMetadata(sourceBucketName,sourceKey);

long partSize = 1024 * 1024 * 100;
// Obtains the size of the object to be copied
long contentLength = objectMetadata.getContentLength();

// Calculates the number of parts
int partCount = (int) (contentLength / partSize);
if (contentLength % partSize != 0) {
    partCount++;
}
System.out.println("total part count:" + partCount);
List<PartETag> partETags = new ArrayList<PartETag>();

long startTime = System.currentTimeMillis();
for (int i = 0; i < partCount; i++) {
    System.out.println("now begin to copy part:" + (i+1));
    long skipBytes = partSize * i;
// Calculates the size of each part
    long size = partSize < contentLength - skipBytes ? partSize : contentLength - skipBytes;
// Creates an UploadPartCopyRequest and performs multipart upload
    UploadPartCopyRequest uploadPartCopyRequest = new UploadPartCopyRequest();
    uploadPartCopyRequest.setSourceKey("/" + sourceBucketName + "/" + sourceKey);
    uploadPartCopyRequest.setBucketName(targetBucketName);
    uploadPartCopyRequest.setKey(targetKey);
    uploadPartCopyRequest.setUploadId(uploadId);
    uploadPartCopyRequest.setPartSize(size);
    uploadPartCopyRequest.setBeginIndex(skipBytes);
    uploadPartCopyRequest.setPartNumber(i + 1);
    UploadPartCopyResult uploadPartCopyResult = client.uploadPartCopy(uploadPartCopyRequest);
// Saves the returned PartETags to the List.
     partETags.add(uploadPartCopyResult.getPartETag());
     System.out.println("now end to copy part:" + (i+1));
}
```

The above program calls the uploadPartCopy method to copy each part. The requirements are basically the same as for UploadPart. You must use setBeginIndex to locate the position corresponding to the start of the next part to upload. You must also specify the object to copy with setSourceKey

Completing Multipart Uploads

Use the code below to complete a multipart upload:

```
CompleteMultipartUploadRequest completeMultipartUploadRequest =
        new CompleteMultipartUploadRequest(bucketName,key, initiateMultipartUploadResult.getUploadId(),
partETags);

// Completes multipart upload
CompleteMultipartUploadResult completeMultipartUploadResult =
        client.completeMultipartUpload(completeMultipartUploadRequest);

// Prints the object's ETag
System.out.println(completeMultipartUploadResult.getETag());
```

In the code above, the partETags are saved in the partETag list during multipart upload. After OSS receives the part list submitted by the users, it will verify the validity of each data part individually. After all the data parts have been verified, OSS will combine these parts into a complete object.

Canceling Multipart Upload Tasks

We can use the abortMultipartUpload method to cancel multipart upload tasks.

```
AbortMultipartUploadRequest abortMultipartUploadRequest =
            new AbortMultipartUploadRequest(bucketName, key, uploadId);

// Cancels the multipart upload
client.abortMultipartUpload(abortMultipartUploadRequest);
```

Getting All Multipart Upload Tasks in the Bucket

We can use the listMultipartUploads method to retrieve all upload tasks in the bucket.

```
// Gets all upload tasks in the bucket
ListMultipartUploadsRequest listMultipartUploadsRequest=new ListMultipartUploadsRequest(bucketName);
MultipartUploadListing listing = client.listMultipartUploads(listMultipartUploadsRequest);

// Traverses all upload tasks
for (MultipartUpload multipartUpload : listing.getMultipartUploads()) {
    System.out.println("Key: " + multipartUpload.getKey() + " UploadId: " + multipartUpload.getUploadId());
}
```

NOTE: Under normal conditions, if a bucket contains more than 1000 multipart upload tasks, the first 1000 will be returned and the IsTruncated parameter in the returned results will be false. The returned NextKeyMarker and NextUploadMarker can be used as the next start point to continue reading the data. If the user wishes to increase the number of multipart upload tasks returned, he can modify the MaxUploads parameter or use the KeyMarker and UploadIdMarker parameters for segmented reading.

Getting Information for All Uploaded Parts

We can use the listParts method to retrieve all the uploaded parts of an upload task.

```
ListPartsRequest listPartsRequest = new ListPartsRequest(bucketName, key, uploadId);

// Gets information for all uploaded parts
PartListing partListing = client.listParts(listPartsRequest);

// Traverses all parts
for (PartSummary part : partListing.getParts()) {
    System.out.println("PartNumber: " + part.getPartNumber() + " ETag: " + part.getETag());
}
```

NOTE: Under normal conditions, if a bucket contains more than 1000 multipart upload tasks, the first 1000 will be returned and the IsTruncated parameter in the returned results will be false. The returned NextPartNumberMarker can be used as the next start point to continue reading the data. If the user wishes to increase the number of upload tasks returned, he can modify the MaxParts parameter or use the PartNumberMarker parameters for segmented reading.

Anti-leech Settings

The OSS collects service fees based on use. To prevent users' data on OSS from being leeched, OSS supports anti-leech based on the field referer in HTTP header.

Setting the Referer White List

We can use the following code to set the Referer white list:

```
OSSClient client = new OSSClient(endpoint, accessId, accessKey);

List<String> refererList = new ArrayList<String>();
// Adds referer
refererList.add("http://www.aliyun.com");
refererList.add("http://www.*.com");
refererList.add("http://www.?.aliyuncs.com");
// Allows the referer field to be blank and sets the Bucket Referer List
BucketReferer br = new BucketReferer(true, refererList);
client.setBucketReferer(bucketName, br);
```

The referer field supports the wildcards "*" and "?". For detailed rule configuration, refer to the product documentation OSS Anti-leech

Retrieving the Referer White List

```
// Retrieves the Bucket Referer List
br = client.getBucketReferer(bucketName);
refererList = br.getRefererList();
for (String referer : refererList) {
    System.out.println(referer);
}


Sample output results:
http://www.aliyun.com
http://www.*.com"
http://www.?.aliyuncs.com
```

Clearing the Referer White List

The Referer white list cannot be cleared directly. You can only reset it to overwrite the previous rules.

```
OSSClient client = new OSSClient(endpoint, accessId, accessKey);
// Allows the referer field and the referer white list name to be blank by default
BucketReferer br = new BucketReferer();
client.setBucketReferer(bucketName, br);
```

Lifecycle Management

OSS provides the object lifecycle management capability to manage objects for users. The user can configure the lifecycle of a bucket to define various rules for the bucket's objects. Currently, users can use rules to delete matched objects. Each rule is composed of the following parts:

- The object name prefix; this rule will only apply to objects with the matched prefix.
- Operation; the operation the user wishes to perform on the matched objects.
- Date or number of days; the user will execute the operation on the objects on the specified date or a specified number of days after the object's last modification time.

Setting Lifecycles

The lifecycle configuration rules are expressed by an xml segment.

```
<LifecycleConfiguration>
  <Rule>
    <ID>delete obsoleted files</ID>
```

```
    <Prefix>obsoleted/</Prefix>
    <Status>Enabled</Status>
    <Expiration>
      <Days>3</Days>
    </Expiration>
  </Rule>
  <Rule>
    <ID>delete temporary files</ID>
    <Prefix>temporary/</Prefix>
    <Status>Enabled</Status>
    <Expiration>
      <Date>2022-10-12T00:00:00.000Z</Date>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

A single lifecycle Config can contain up to 1000 rules.

Explanations of each field:

- The ID field is used to uniquely identify a rule (inclusion relations, such as abc and abcd, cannot exist between IDs).
- Prefix indicates the rules used for objects in the bucket with the specified prefix.
- Status indicates the status of this rule. The statuses are Enabled and Disabled, indicating if the rule is enabled or disabled.
- In the Expiration node, Days indicates that an object will be deleted a specified number of days after its last modification. Date indicates that objects will be deleted after the specified absolute time (the absolute time follows the ISO8601 format).

Using the following code, we can set the above lifecycle rules.

```
OSSClient client = new OSSClient(endpoint, accessId, accessKey);

SetBucketLifecycleRequest req = new SetBucketLifecycleRequest(bucketName);
// Adds a Lifecycle rule
req.AddLifecycleRule(new LifecycleRule("delete obsoleted files", "obsoleted/", RuleStatus.Enabled, 3));
req.AddLifecycleRule(new LifecycleRule("delete temporary files", "temporary/", RuleStatus.Enabled,
        DateUtil.parseIso8601Date("2022-10-12T00:00:00.000Z")));
// Sets Bucket Lifecycle
client.setBucketLifecycle(req);
```

We can use the following code to retrieve the above lifecycle rules.

```
OSSClient client = new OSSClient(endpoint, accessId, accessKey);

// Retrieves the above Bucket Lifecycle
List<LifecycleRule> rules = client.getBucketLifecycle(bucketName);
Assert.assertEquals(rules.size(), 2);

System.out.println("Rule1: ");
LifecycleRule r1 = rules.get(0);
```

```
System.out.println("ID: " + r1.getId());
System.out.println("Prefix: " + r1.getPrefix());
System.out.println("Status: " + r1.getStatus().toString());
System.out.println("ExpirationDays: " + r1.getExpriationDays());
System.out.println();

System.out.println("Rule2: ");
LifecycleRule r2 = rules.get(1);
System.out.println("ID: " + r2.getId());
System.out.println("Prefix: " + r2.getPrefix());
System.out.println("Status: " + r2.getStatus().toString());
System.out.println("ExpirationTime: " + DateUtil.formatIso8601Date(r2.getExpirationTime()));
```

Using the following code, we can delete the lifecycle rules in a bucket.

```
OSSClient client = new OSSClient(endpoint, accessId, accessKey);

client.deleteBucketLifecycle(bucketName);
```

Authorized Access

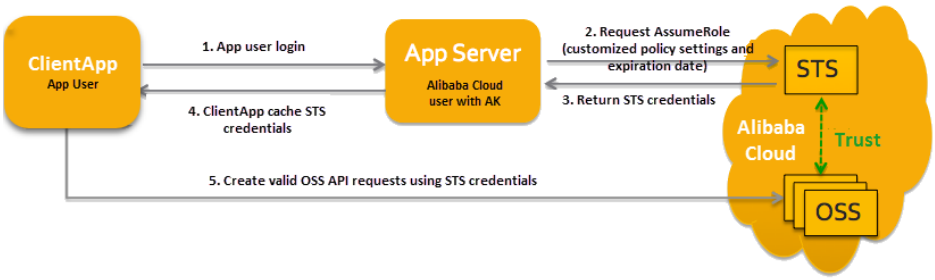
Using STS Service Temporary Authorization

Introduction

Through the AliCloud STS service, OSS can temporarily grant authorized access.AliCloud STS is a web service that provides a temporary access token to a cloud computing user. Using STS, you can grant access credentials to a third-party application or federated user (you can manage the user IDs) with customized permissions and validity periods. Third-party applications or federated users can use these access credentials to directly call the AliCloud product APIs or use the SDKs provided by AliCloud products to access the cloud product APIs.

- You do not need to expose you long-term key (AccessKey) to a third-party application and only need to generate an access token and send the access token to the third-party application. You can customize the access permission and validity of this token.
- You do not need to care about permission revocation issues. The access credential automatically becomes invalid when it expires.

Using an App as an example, the interaction process is shown below:



The solution is described in detail as follows:

1. Log in as the app user. App user IDs are managed by the client. Clients can customize the ID management system and may also use external Web accounts or OpenID. For each valid app user, the AppServer can precisely define the minimum access permission.
2. The AppServer requests a security token from the STS. Before calling STS, the AppServer needs to determine the minimum access permission (described in policy syntax) of app users and the expiration time of the authorization.Then, the security token is obtained by calling the STS' AssumeRole interface.
3. The STS returns a valid access credential to the AppServer, where the access credential includes a security token, a temporary access key (AccessKeyId and AccessKeySecret), and the expiry time.
4. The AppServer returns the access credential to the ClientApp. The ClientApp caches this credential. When the credential becomes invalid, the ClientApp needs to request a new valid access credential from the AppServer. For example, if the access credential is valid for one hour, the ClientApp can request the AppServer to update the access credential every 30 minutes.
5. The ClientApp uses the access credential cached locally to request for AliCloud Service APIs. The ECS is aware of the STS access credential, relies on STS to verify the credential, and correctly responds to the user request.

Using STS Credentials to Construct Signed Requests

After obtaining the STS temporary credential, the user' s client generates an OSSClient using the contained security token and temporary access key (AccessKeyId, AccessKeySecret). Using an object upload as an example:

```
String accessKeyId = "<accessKeyId>";
String accessKeySecret = "<accessKeySecret>";
String securityToken = "<securityToken>"
// Uses Hangzhou as an example
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";

OSSClient client = new OSSClient(endpoint, accessKeyId, accessKeySecret,securityToken);
```

Using URL Signature to Authorize Access

Generating a Signed URL

You can provide users with a temporary access URL by generating a signed URL. During URL generation, you can specify the URL expiration time to limit the duration of the user’ s access.

Generating a Signed URL

The code is as follows:

```
String bucketName = "your-bucket-name";
String key = "your-object-key";

// Sets the URL expiration time to 1 hour
Date expiration = new Date(new Date().getTime() + 3600 * 1000);

// Generates the URL
URL url = client.generatePresignedUrl(bucketName, key, expiration);
```

Generated URLs use the GET access method by default. This way, users can directly use a browser to access the relevant content.

Generating Other HTTP Method URLs

For users to temporarily use other operations (e.g. uploading or deleting objects), you may have to sign a URL for another method. For example:

```
// Generates a PUT method URL
URL url = client.generatePresignedUrl(bucketName, key, expiration, HttpMethod.PUT);
```

By importing the HttpMethod.PUT parameter, users can use generated URLs to upload objects.

Adding User-defined Parameters (UserMetadata)

If you want to generate a signed URL to upload objects and specify the information for UserMetadata, Content-Type, and other headers, you can proceed as follows:

```
// Creates request
GeneratePresignedUrlRequest generatePresignedUrlRequest = new GeneratePresignedUrlRequest(bucketName,
key);

// Sets HttpMethod to PUT
generatePresignedUrlRequest.setMethod(HttpMethod.PUT);

// Adds UserMetadata
generatePresignedUrlRequest.addUserMetadata("author", "baymax");

// Adds Content-Type
request.setContentType("application/octet-stream");
```

```
// Generates signed URL
URL url = client.generatePresignedUrl(generatePresignedUrlRequest);
```

Please note that the above process only generates a signed URL. You still must add the meta information in the request header. You can refer to the following code.

Using Signed URLs to Send Requests

Currently, the Java SDK supports the put object and get object URL signature requests.

Using the getobject URL Signature Method

```
//The server generates the URL signature string
OSSClient Server = new OSSClient(endpoint, accessId, accessKey);
Date expiration = DateUtil.parseRfc822Date("Wed, 18 Mar 2015 14:20:00 GMT");
GeneratePresignedUrlRequest request = new GeneratePresignedUrlRequest(bucketName, key, HttpMethod.GET);
//Sets the expiration time
request.setExpiration(expiration);
// Generates the URL signature (HTTP GET request)
URL signedUrl = Server .generatePresignedUrl(request);
System.out.println("signed url for getObject: " + signedUrl);

//The client uses the URL signature string to send the request
OSSClient client = new OSSClient(endpoint, "" , "" );
Map<String, String> customHeaders = new HashMap<String, String>();
// Adds the GetObject request header
customHeaders.put("Range", "bytes=100-1000");
OSSObject object = client.getObject(signedUrl,customHeaders);
```

Using the putobject URL Signature Method

```
//The server generates the URL signature string
OSSClient Server = new OSSClient(endpoint, accessId, accessKey);
Date expiration = DateUtil.parseRfc822Date("Wed, 18 Mar 2015 14:20:00 GMT");
GeneratePresignedUrlRequest request = new GeneratePresignedUrlRequest(bucketName, key, HttpMethod.PUT);
//Sets the expiration time
request.setExpiration(expiration);
//Sets Content-Type
request.setContentType("application/octet-stream");
// Adds User Meta
request.addUserMetadata("author", "aliy");
// Generates the URL signature (HTTP PUT request)
URL signedUrl = Server.generatePresignedUrl(request);
System.out.println("signed url for putObject: " + signedUrl);

//The client uses the URL signature string to send the request
OSSClient client = new OSSClient(endpoint, "" , "" );
File f = new File(filePath);
FileInputStream fin = new FileInputStream(f);
// Adds the PutObject request header
Map<String, String> customHeaders = new HashMap<String, String>();
```

```
customHeaders.put("Content-Type", "application/octet-stream");
// Adds User Meta
customHeaders.put("x-oss-meta-author", "aliy");
PutObjectResult result = client.putObject(signedUrl, fin, f.length(), customHeaders);
```

Cross-Origin Resource Sharing (CORS)

CORS allows web applications to access resources in other regions. OSS provides an interface to allow developers to easily control cross-origin access permissions.

Setting CORS Rules

Using the setBucketCORS method, we can set a CORS rule for a specified bucket. If an original rule exists, it will be overwritten. Parameters for specific rules are generally set through CORSRule. The code is as follows:

```
SetBucketCORSRequest request = new SetBucketCORSRequest();
request.setBucketName(bucketName);
ArrayList<CORSRule> putCorsRules = new ArrayList<CORSRule>();
//The CORS rule container, each bucket allows up to 10 rules

CORSRule corRule = new CORSRule();
ArrayList<String> allowedOrigin = new ArrayList<String>();
//Specifies allowed cross-origin request origins
allowedOrigin.add( "http://www.b.com");
ArrayList<String> allowedMethod = new ArrayList<String>();
//Specifies the allowed cross-origin request methods (GET/PUT/DELETE/POST/HEAD)
allowedMethod.add("GET");
ArrayList<String> allowedHeader = new ArrayList<String>();
//Controls if the headers specified in the OPTIONS' prefetch command's Access-Control-Request-Headers are
allowed.
allowedHeader.add("x-oss-test");
ArrayList<String> exposedHeader = new ArrayList<String>();
//Specifies the response headers users are allowed to access from the application
exposedHeader.add("x-oss-test1");
corRule.setAllowedMethods(allowedMethod);
corRule.setAllowedOrigins(allowedOrigin);
corRule.setAllowedHeaders(allowedHeader);
corRule.setExposeHeaders(exposedHeader);
// Specifies the cache time for the returned results of browser prefetch (OPTIONS) requests to a specific resource,
unit: seconds.
corRule.setMaxAgeSeconds(10);
//Maximum of 10 rules allowed
putCorsRules.add(corRule);
request.setCorsRules(putCorsRules);
oss.setBucketCORS(request);
```

Here, you must note the following:

- Each Bucket allows up to 10 rules.
- The AllowedOrigins and AllowedMethods each supports up to one "*" wildcard. "*" expresses that all origin regions or operations satisfy the condition. However, AllowedHeaders and ExposeHeaders do not support wildcards.

Retrieving CORS Rules

We can refer to the bucket's CORS rules through the getBucketCORSRules method. The code is as follows:

```
ArrayList<CORSRule> corsRules;
//Retrieves list of CORS rules
corsRules =  (ArrayList<CORSRule>) oss.getBucketCORSRules(bucketName);
for (CORSRule rule : corsRules) {
for (String allowedOrigin1 : rule.getAllowedOrigins()) {
   //Retrieves allowed cross-origin request origins
      System.out.println(allowedOrigin1);
   }
for (String allowedMethod1 : rule.getAllowedMethods()) {
   //Retrieves allowed cross-origin request methods
      System.out.println(allowedMethod1);
   }

   if (rule.getAllowedHeaders().size() > 0){
      for (String allowedHeader1 : rule.getAllowedHeaders()){
         //Retrieves the list of allowed headers
         System.out.println(allowedHeader1);
      }
   }

   if (rule.getExposeHeaders().size() > 0){
      for (String exposeHeader : rule.getExposeHeaders()){
         //Retrieves allowed headers
         System.out.println(exposeHeader);
      }
   }

   if ( null != rule.getMaxAgeSeconds()){
      System.out.println(rule.getMaxAgeSeconds());
   }
}
```

Deleting CORS Rules

Use this to disable CORS for the specified bucket and clear all rules.

```
// Clears the CORS rules in the bucket
```

```
oss.deleteBucketCORSRules(bucketName);
```

In the same way, only the bucket owner can delete rules.

Exceptions

The OSS Java SDK has two exceptions: ClientException and OSSException. Both are derived, directly or indirectly, from RuntimeException.

ClientException

ClientException indicates an internal SDK exception, such as no set BucketName, cannot connect to the network, etc.

OSSException

OSSException indicates a server error, which is generated by parsing a server error message. OSSExceptions generally have the following components:

- Code: the error code OSS returns to users.
- Message: the detailed error message provided by OSS.
- RequestId: the UUID that uniquely identifies the request. When you cannot solve the problem, you can seek help from OSS development engineers by providing this RequestId.
- HostId: used to identify the accessed OSS cluster (currently unified as oss.aliyuncs.com)

The following are common OSS exceptions:

Error Code	Description
AccessDenied	Access denied
BucketAlreadyExists	The bucket already exists
BucketNotEmpty	The bucket is not empty
EntityTooLarge	The entity is too large
EntityTooSmall	The entity is too small
FileGroupTooLarge	The file group is too large
FilePartNotExist	A file part does not exist
FilePartStale	A file part has expired
InvalidArgument	Parameter format error
InvalidAccessKeyId	The Access Key ID does not exist

InvalidBucketName	The bucket name is invalid
InvalidDigest	The digest is invalid
InvalidObjectName	The object name is invalid
InvalidPart	A part is invalid
InvalidPartOrder	The part order is invalid
InvalidTargetBucketForLogging	The logging operation has an invalid target bucket
InternalError	Internal OSS error
MalformedXML	Illegal XML format
MethodNotAllowed	The method is not supported
MissingArgument	A parameter is missing
MissingContentLength	The content length is missing
NoSuchBucket	The bucket does not exist
NoSuchKey	The file does not exist
NoSuchUpload	The Multipart Upload ID does not exist
NotImplemented	The method cannot be processed
PreconditionFailed	Preprocessing error
RequestTimeTooSkewed	The request initiation time exceeds the server time by 15 minutes
RequestTimeout	Request timed out
SignatureDoesNotMatch	Signature error
TooManyBuckets	The user's bucket quantity exceeds the limit

Python-SDK

Installation

The following introduces the complete process and demonstrates how to use for basic OSS bucket and object operations on Windows and Linux platforms.

Environment Requirements

The following introduces the complete process and demonstrates how to use for basic OSS bucket and object operations on Windows and Linux platforms.

After Python is installed:

- Input Python in Linux shell and click the enter key to view the Python version. As shown below:

```
Python 2.5.4 (r254:67916, Mar 10 2010, 22:43:17)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-46)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Input Python in the Windows cmd environment and click the enter key to view the Python version. As shown below:

```
C:\Documents and Settings\Administrator>python
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

As shown above, Python has been installed successfully.

- Exceptions. E.g.: after entering python in the Windows cmd environment and clicking Enter, the system prompts “Not an internal or external command”. In such a case, check the configuration “Environment variables” - “Path” and add the Python installation path.

If Python is not installed, you can get its installation package from Python Official Website. The website provides detailed instructions and guidance for installing and using Python.

Installing and Verifying SDK

The following teaches you how to install the Python SDK on Windows and Linux platforms, as well as how to verify that it has been installed successfully.

Installing the SDK

1. Open your browser and enter oss.aliyun.com.
2. In “Product Help” - “Developer Resources”, find the Python SDK link.
3. Click on the link and choose to save the SDK installation package.
4. After downloading it, you can get the installation package, with a name similar to OSS_Python_API_xxxxxxxx.tar.gz.
5. Go to the installation package’s directory and decompress the tar.gz package. Linux can use the tar -zxvf OSS_Python_API_xxxxxxxx.tar.gz command to decompress the package.

Windows can use 7-Zip or another decompression tool.

6. After decompression, obtain the files and directories shown below

```
README
OSS_Python_SDK.pdf
setup.py
osscmd
oss/
oss_api.py
oss_util.py
oss_xml_handler.py
oss_sample.py
```

7. SDK installation for two platforms
   - Let us assume that the SDK has been decompressed on disk D on the Windows platform. The install log will be as follows:

```
D:\>cd  OSS_Python_API_20130712
D:\ OSS_Python_API_20130712 >python setup.py install
running install
running build
running build_py
creating build
creating build\lib
creating build\lib\oss
copying oss\oss_api.py -> build\lib\oss
copying oss\oss_sample.py -> build\lib\oss
copying oss\oss_util.py -> build\lib\oss
copying oss\oss_xml_handler.py -> build\lib\oss
copying oss\pkg_info.py -> build\lib\oss
copying oss\__init__.py -> build\lib\oss
running install_lib
creating C:\Python27\Lib\site-packages\oss
copying build\lib\oss\oss_api.py -> C:\Python27\Lib\site-packages\oss
copying build\lib\oss\oss_sample.py -> C:\Python27\Lib\site-packages\oss
copying build\lib\oss\oss_util.py -> C:\Python27\Lib\site-packages\oss
copying build\lib\oss\oss_xml_handler.py -> C:\Python27\Lib\site-packages\oss
copying build\lib\oss\pkg_info.py -> C:\Python27\Lib\site-packages\oss
copying build\lib\oss\__init__.py -> C:\Python27\Lib\site-packages\oss
byte-compiling C:\Python27\Lib\site-packages\oss\oss_api.py to oss_api.pyc
byte-compiling C:\Python27\Lib\site-packages\oss\oss_sample.py to oss_sample.pyc
byte-compiling C:\Python27\Lib\site-packages\oss\oss_util.py to oss_util.pyc
byte-compiling C:\Python27\Lib\site-packages\oss\oss_xml_handler.py to oss_xml_handler.pyc
byte-compiling C:\Python27\Lib\site-packages\oss\pkg_info.py to pkg_info.pyc
byte-compiling C:\Python27\Lib\site-packages\oss\__init__.py to __init__.pyc
running install_egg_info
Writing C:\Python27\Lib\site-packages\oss-0.1.3-py2.7.egg-info
```

   - Let us assume that the SDK has been decompressed in the OSS directory on the Linux platform. The install log will be as follows:

```
[oss@oss python]$ sudo python setup.py install
Password:
running install
running bdist_egg
running egg_info
writing oss.egg-info/PKG-INFO
writing top-level names to oss.egg-info/top_level.txt
writing dependency_links to oss.egg-info/dependency_links.txt
writing manifest file 'oss.egg-info/SOURCES.txt'
installing library code to build/bdist.linux-x86_64/egg
running install_lib
running build_py
creating build/bdist.linux-x86_64/egg
creating build/bdist.linux-x86_64/egg/oss
copying build/lib/oss/oss_util.py -> build/bdist.linux-x86_64/egg/oss
copying build/lib/oss/__init__.py -> build/bdist.linux-x86_64/egg/oss
copying build/lib/oss/pkg_info.py -> build/bdist.linux-x86_64/egg/oss
copying build/lib/oss/oss_xml_handler.py -> build/bdist.linux-x86_64/egg/oss
copying build/lib/oss/oss_sample.py -> build/bdist.linux-x86_64/egg/oss
copying build/lib/oss/oss_api.py -> build/bdist.linux-x86_64/egg/oss
byte-compiling build/bdist.linux-x86_64/egg/oss/oss_util.py to oss_util.pyc
byte-compiling build/bdist.linux-x86_64/egg/oss/__init__.py to __init__.pyc
byte-compiling build/bdist.linux-x86_64/egg/oss/pkg_info.py to pkg_info.pyc
byte-compiling build/bdist.linux-x86_64/egg/oss/oss_xml_handler.py to oss_xml_handler.pyc
byte-compiling build/bdist.linux-x86_64/egg/oss/oss_sample.py to oss_sample.pyc
byte-compiling build/bdist.linux-x86_64/egg/oss/oss_api.py to oss_api.pyc
creating build/bdist.linux-x86_64/egg/EGG-INFO
copying oss.egg-info/PKG-INFO -> build/bdist.linux-x86_64/egg/EGG-INFO
copying oss.egg-info/SOURCES.txt -> build/bdist.linux-x86_64/egg/EGG-INFO
copying oss.egg-info/dependency_links.txt -> build/bdist.linux-x86_64/egg/EGG-INFO
copying oss.egg-info/top_level.txt -> build/bdist.linux-x86_64/egg/EGG-INFO
zip_safe flag not set; analyzing archive contents...
oss.oss_sample: module references __file__
creating 'dist/oss-0.1.3-py2.5.egg' and adding 'build/bdist.linux-x86_64/egg' to it
removing 'build/bdist.linux-x86_64/egg' (and everything under it)
Processing oss-0.1.3-py2.5.egg
removing '/usr/ali/lib/python2.5/site-packages/oss-0.1.3-py2.5.egg' (and everything under it)
creating /usr/ali/lib/python2.5/site-packages/oss-0.1.3-py2.5.egg
Extracting oss-0.1.3-py2.5.egg to /usr/ali/lib/python2.5/site-packages
oss 0.1.3 is already the active version in easy-install.pth
Installed /usr/ali/lib/python2.5/site-packages/oss-0.1.3-py2.5.egg
Processing dependencies for oss==0.1.3
Finished processing dependencies for oss==0.1.3
```

8. Uninstall SDK.

   The SDK will be installed in different directories on different machines and platforms. To uninstall it, you must go to the SDK install directory and delete the relevant installed directories. For example, the ‘/usr/ali/lib/python2.5/site-packages/oss-0.1.3-py2.5.egg’ directory shown in the previous step.You also can use the following command to find the relevant installed directories in the site-packages directory of the current Python version and delete them to uninstall the SDK.

```
>>> import sys
>>> print sys.path
['','/usr/ali/lib/python2.5/site-packages/httplib2-0.7.7-py2.5.egg', '/usr/ali/lib/python2.5/site-packages/oss-0.1.3-py2.5.egg','/usr/ali/lib/python25.zip','/usr/ali/lib/python2.5', '/usr/ali/lib/python2.5/plat-linux2','/usr/ali/lib/python2.5/lib-tk','/usr/ali/lib/python2.5/lib-dynload', '/usr/ali/lib/python2.5/site-packages']
```

Preface

Introduction

This document introduces the installation and use of the OSS Python SDK (for version 0.4.2 in particular).This document assumes that you have already subscribed to the AliCloud OSS service and created an Access Key ID and Access Key Secret.In the document, ID represents the Access Key ID and KEY indicates the Access Key Secret. If you have not yet subscribed to or do not know about the OSS service, please log into the OSS Product Homepage for more help.

Version Revisions

Python SDK Development Kit (2015-07-07) Version 0.4.0

Updates:

- Supported STS function in osscmd

Python SDK Development Kit (2015-06-24) Version 0.3.9

Updates:

- Added copylargefile command in osscmd to support the copying of large files

Python SDK Development Kit (2015-04-13) Version 0.3.8

Updates:

- Fixed the invalid max_part_num specified for multiupload problem in osscmd
- Added an md5 check for the part specified by upload_part in oss_api

Python SDK Development Kit (2015-01-29) Version 0.3.7

Updates:

- Added the referer and lifecycle interfaces in oss_api.
- Added referer and lifecycle commands in osscmd.
- Fixed the invalid upload_id problem in osscmd.

Python SDK Development Kit (2014-12-31) Version 0.3.6

Updates:

- Added the check_point function for the osscmd uploadfromdir command, using the --check_point optional setting.
- Added the --force function for the osscmd deleteallobject command, force deleting all files.
- Added the --thread_num option in osscmd's multipart and uploadfromdir/downloadtodir commands, allowing users to adjust the number of threads.
- Added the file name-based Content-Type generation function in oss_api.
- Added the --temp_dir option for the osscmd downloadtodir command, supporting temporarily saving the downloaded file to the specified directory.
- Added the --check_md5 option for osscmd, allowing md5 checks on upload files.

For a Quick Start Guide, refer to the README file in the SDK

Python SDK Development Kit (2014-05-09)

Updates:

- Fixed the logger initialization error bug in oss_util.
- Optimized the multi_upload_file upload interface in oss_api for certain situations, reducing the number of re-uploads due to network exceptions.

Quick Start

In this chapter, you will learn how to use the basic functions of the OSS Python SDK.

Step-1. Initializing an OssAPI

SDK OSS operations are performed through the OssAPI class. The code below creates an OssAPI object:

```
from oss.oss_api import *
#Using Hangzhou as an example
endpoint=” oss-hangzhou.aliyuncs.com”
accessKeyId, accessKeySecret=” your id” ,” your secret”
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
```

In the above code, the variables accessKeyId and accessKeySecret are allocated to the user by the system. They are called the ID pair and used to identify the user. They are used to perform signature verification when accessing OSS. For a detailed introduction of the OssAPI class, refer to OssAPI.

Step-2. Creating Buckets

Buckets are the OSS global namespace. They are equivalent to a data container and can store numerous objects. You can create a bucket with the following code:

```
from oss.oss_api import *
#Using Hangzhou as an example
endpoint=” oss-hangzhou.aliyuncs.com”
accessKeyId, accessKeySecret=” your id” ,” your secret”
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Setting the bucket permission to private
res = oss.create_bucket(bucket,"private")
print "%s\n%s" % (res.status, res.read())
```

Call the create_bucket method to return an HTTP Response class. For bucket naming rules, refer to [Naming Rules] in Bucket.

Step-3. Uploading objects

Objects are the basic data elements in OSS. You can simply think of them as files. The code below will upload an object:

```
from oss.oss_api import *

oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
res = oss.put_object_from_file(bucket, object, "test.txt")
print "%s\n%s" % (res.status, res.getheaders())
```

For object naming rules, refer to the naming rules in Object. For more information on uploading objects, refer to [Uploading Objects] in Object.

Step-4. Listing all objects

When you complete a series of uploads, you may need to view which objects are in a bucket. This can be done with the following program:

```
from oss.oss_api import *

oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
res = oss.get_bucket("bucketname")
print "%s\n%s" % (res.status, res.read())
```

For more flexible parameter configurations, refer to the [List Bucket Objects] in Object.

Step-5. Retrieving a specified object

You can refer to the code below to easily retrieve an object:

```
from oss.oss_api import *

oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Directly reading the object to a local file
res = oss.get_object_to_file(bucket, object, "/filepath/test.txt")
print "%s\n%s" % (res.status, res.getheaders())
```

You can read the object meta information from the response header.

Bucket

OSS uses buckets as the namespaces of user files and also as the management objects for advanced functions such as charging, permission control, and log recording. The bucket name must be globally unique in the entire OSS and cannot be changed. Every object stored on the OSS must be included in a bucket. One application, such as an image sharing website, can correspond to one or more buckets. A user can create a maximum of 10 buckets, but there is no limit on the number of objects in each bucket. Each bucket can store up to 2 PB of data.

Naming Rules

The bucket naming rules are as follows:

- It can only contain lower-case letters, numbers, and dashes (-).
- It must start with a lower-case letter or number.
- The length must be 3-63 bytes

Creating Buckets

```
from oss.oss_api import *
#Using Hangzhou as an example
endpoint=” oss-hangzhou.aliyuncs.com”
accessKeyId, accessKeySecret=” your id” ,” your secret”
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)

res = oss.create_bucket("bucketname","private") #Sets the bucket permission to private
print "%s\n%s" % (res.status, res.read())
```

To create a bucket you must enter the bucket name and ACL.Because bucket names are globally unique, do your best to ensure your bucket names are not the same as other people’ s.ACL currently supports the values private, public-read, and public-read-write. For information on permissions, please refer to OSS Access Control

Listing all Buckets of a User

This can be performed using the get_service or list_all_my_buckets methods. The two methods are equivalent.

```
from oss.oss_api import *
from oss import oss_xml_handler
#Using Hangzhou as an example
endpoint=” oss-hangzhou.aliyuncs.com”
accessKeyId, accessKeySecret=” your id” ,” your secret”
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)

res = oss.list_all_my_buckets()
buckets_xml=oss_xml_handler.GetServiceXml(res.read())
for bucket_info in buckets_xml.bucket_list:
print "----------------------------------------"
print "Location:"+bucket_info.location
print "Name:"+bucket_info.name
print "CreationDate:"+bucket_info.creation_date
```

All bucket information exists in xml format in the HTTP response’ s body. It can be parsed using GetServiceXml in oss_xml_handler.

CNAME Access

After a user directs his own domain name’ s CNAME to the domain name of one of his buckets, he can access OSS through his domain name:

```
from oss.oss_api import *
#Entering an endpoint as your bound domain name
endpoint=” cname.com”
accessKeyId, accessKeySecret=” your id” ,” your secret”
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Uploading content to the bucket CNAME directs to
res=oss.put_object_from_string( “bucketname” ,” objectname” ,” conetnet” )
```

Users just need to change the endpoint originally expected to be entered in the bucket to the CNAME domain name when creating an OssAPI instance. At the same time, users must note that when using this OssAPI instance for subsequent operations, they can only operate on the bucket the CNAME directs to.

Setting Bucket ACL

When using the create_bucket or put_bucket method, if this bucket exists and belongs to the request initiator, the request’ s ACL settings will overwrite the original settings, thus setting the Bucket ACL.

```
from oss.oss_api import *
from oss import oss_xml_handler
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)

#Setting the bucket as already existing; sets the bucket acl as public-read
res = oss.create_bucket(bucket,"public-read")
print "%s\n%s" % (res.status, res.read())
```

ACL currently supports the values private, public-read, and public-read-write. For information on permissions, please refer to OSS Access Control

Retrieving Bucket ACL

get_bucket_acl is used to obtain the ACL for this bucket

```
from oss.oss_api import *
from oss import oss_xml_handler
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Retrieving Bucket ACL
res= oss.get_bucket_acl(bucket)
acl_xml=oss_xml_handler.GetBucketAclXml(res.read())
print acl_xml.grant
```

ACL information exists in xml format in the HTTP response’ s body. The results are obtained by parsing the required xml content using the GetBucketAclXml class in oss_xml_handler.

Retrieving Bucket Addresses

get_bucket_location is used to obtain the location for this bucket

```
from oss.oss_api import *
from oss import oss_xml_handler
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Retrieving Bucket Addresses
res= oss.get_bucket_acl(bucket)
location_xml=oss_xml_handler.GetBucketLocationXml(res.read())
print location_xml.location
```

Location information exists in xml format in the HTTP response’ s body. It can be parsed using the GetBucketLocationXml class in oss_xml_handler.

Deleting Buckets

```
from oss.oss_api import *
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)

res = oss. delete_bucket(bucket)
print "%s\n%s" % (res.status, res.read())
```

Please note that if the bucket is not empty (i.e., bucket contains objects or multipart upload fragments), it cannot be deleted. You must delete all objects and fragments in a bucket before deleting the bucket.

OssAPI

The OssAPI class provides methods for operating on buckets and objects. Users can call these methods to operate on OSS. Most of the interfaces provided in OssAPI directly return HTTP responses. For HTTP response definitions, please refer to the descriptions in the official Python documentation. In the example, we use res to represent an HTTP response instance. res.status represents the OSS HTTP Server status code returned after the Python SDK sends an HTTP request to OSS. See the OSS API documentation for the specific status codes. res.getheaders() indicates the OSS HTTP Server response headers. res.read() indicates the HTTP response body. In some situations, the body has no content.

Initializing OssAPI

```
from oss.oss_api import *
#Using Hangzhou as an example
endpoint=" oss-hangzhou.aliyuncs.com"
accessKeyId, accessKeySecret=" your id"," your secret"
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
```

Here, the endpoint must enter the corresponding value of the region of the bucket to be operated on. For relevant endpoint details, refer here.

Configuring OssAPI

To configure detailed OssAPI parameters, you can use the following methods.

Method	Description	Default Value
set_timeout	Sets the timeout time	10
set_debug	Sets the debug mode	True

set_retry_times	Sets the number of retries when an error is encountered	3
set_send_buf_size	Sets the buffer size when sending data	8192
set_recv_buf_size	Sets the buffer size when receiving data	1024*1024*10

Object

In OSS, objects are the basic data units for user operation.The maximum size of a single object may vary depending on the data uploading mode. The size of an object cannot exceed 5 GB in the Put Object mode or 48.8 TB in the multipart upload mode. An object includes the key, meta, and data. The key is the object name; meta is the user's description of the object, composed of a series of name-value pairs; and data is the object data.

Naming Rules

Object naming rules:

- It uses UTF-8 encoding
- The length must be 1-1023 bytes
- It cannot start with "/" or "\"
- It cannot contain "\r" or "\n" line breaks

Uploading Objects

Simple Upload

The two methods below can both upload objects.

```
from oss.oss_api import *
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Directly uploading strings
res=oss.put_object_from_string(bucket,object,"string content")
print "%s\n%s" % (res.status, res.read())


from oss.oss_api import *
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Uploading local files
res=oss.put_object_from_file("bucketname","objectname","test.txt")
print "%s\n%s" % (res.status, res.read())
```

When uploading in this manner, the largest file cannot exceed 5G. For larger files, you can use MultipartUpload.

Creating Simulated Folders

The OSS service does not use folders. All elements are stored as objects. However, users can create simulated folders using the following code:

```
from oss.oss_api import *
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Uploading a blank string and making the object end in "/"
res=oss.put_object_from_string("bucketname","folder_name/","")
print "%s\n%s" % (res.status, res.read())
```

Creating a simulated folder is in fact creating an object with a size of 0. This object can also be uploaded and downloaded. The console will display any object ending with "/" as a folder. Therefore, users can create simulated folders this way. For accessing folders, refer to the [Folder Simulation Function]

Setting the Object's Http Header

The OSS service allows users to customize the object Http Header. The following code sets the expiration time for the object:

```
from oss.oss_api import *
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Setting the expiretime
header={ “Expires” :”  Fri, 28 Feb 2012 05:38:42 GMT” }
#Uploading a local file with a header
res=oss.put_object_from_file(bucket,object,"test.txt",headers= header)
print "%s\n%s" % (res.status, res.read())
```

By importing dic-type parameter headers, you can set the upload header.

You can set the HTTP header to: Cache-Control, Content-Disposition, Content-Encoding, Expires, and Content-MD5. For details on the headers, please see RFC2616. We suggest users add a Content-MD5 value when uploading. In this case, the OSS calculates the body's Content-MD5 and checks if the two are the same.

Setting User Meta

The OSS allows users to define meta information to describe the object. For example:

```
from oss.oss_api import *
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Setting user-meta
```

```
headers={ “x-oss-meta-user” :” baymax” }
#Uploading local files
res=oss.put_object_from_file(bucket, object,"test.txt",headers= headers)
print "%s\n%s" % (res.status, res.read())
```

user meta is header information with the "x-oss-meta" prefix. As with uploading of http header, it is imported using the dic-type headers parameters.A single object can have multiple similar parameters, but the total size of all user meta cannot exceed 2 KB.

> NOTE: The user meta name is not case sensitive. For instance, when a user uploads an object and defines the meta name as "x-oss-meta-USER", the parameter stored in the header will be: "x-oss-meta-user".

Multipart Upload

OSS allows users to split an object into several requests for uploading to the server. Concerning multipart upload content, refer to the Object Multipart Upload section in MultipartUpload.

List Bucket Objects

Listing Objects

```
from oss.oss_api import *
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)

#Listing bucket objects
res= oss.list_bucket(bucket)
objects_xml=oss_xml_handler.GetBucketXml(res.read())
print objects_xml.show()
```

Object information exists in xml format in the HTTP response's body. It must be parsed using the oss_xml_handler.GetBucketXml class.

Extended Parameters

We can use more parameters for more complex functions. For example:

```
from oss.oss_api import *
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)

#Listing bucket CommonPrefixes with the prefix "pic" and ending in "/"
res= oss.list_bucket(bucket,prefix=” pic” ,delimiter=” /” )
objects_xml=oss_xml_handler.GetBucketXml(res.read())
print objects_xml.show()
```

The above code lists bucket CommonPrefixes with the prefix "pic" and ending in "/". For example, "pic-people/". Settable parameter names and their functions:

Name	Function
delimiter	Character used to group object names. All objects whose names contain the specified prefix and that appear between the Delimiter characters for the first time are used as a group of elements: CommonPrefixes.
marker	Sets the returned results to begin from the first entry after the marker in alphabetical order.
maxkeys	Limits the maximum number of objects returned for one request. If not specified, the default value is 100. The maxkeys value cannot exceed 1000.
prefix	Requires the returned object keys to be prefixed with prefix. Note that the keys returned from queries using a prefix will still contain the prefix.

Folder Function Simulation

We can use a combination of Delimiter and Prefix to simulate folder functions. Combinations of Delimiter and Prefix serve the following purposes: Setting Prefix as the name of a folder enumerates the files starting with this prefix, recursively returning all files and subfolders in this folder. When the Delimiter is set as "/", the returned values will enumerate the files in the folder and the subfolders will be returned in the CommonPrefixes section. Recursive files and folders in subfolders will not be displayed. If the bucket contains 4 files: 'oss.jpg', 'fun/test.jpg', 'fun/movie/001.avi', and 'fun/movie/007.avi'. We use the "/" symbol as the separator for folders.

- List bucket files

```
from oss.oss_api import *
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)

#Listing bucket objects
res= oss.list_bucket(bucket)
objects_xml=oss_xml_handler.GetBucketXml(res.read())
#Formatting output results
print "Objects:"
for object_info in objects_xml.content_list:
    print object_info.key

print "CommonPrefixs:"
for prefix in objects_xml.prefix_list:
    print prefix
```

```
Objects:
fun/movie/001.avi
fun/movie/007.avi
fun/test.jpg
oss.jpg

CommonPrefixs:
```

- Recursively list all files in a directory

```
from oss.oss_api import *
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)

#Listing all files in the bucket's "fun/" directory
res= oss.list_bucket(“bucketname”,prefix=”fun/”)
objects_xml=oss_xml_handler.GetBucketXml(res.read())
#Formatting output results
print "Objects:"
for object_info in objects_xml.content_list:
    print object_info.key

print "CommonPrefixs:"
for prefix in objects_xml.prefix_list:
    print prefix


Output:
Objects:
fun/movie/001.avi
fun/movie/007.avi
fun/test.jpg

CommonPrefixs:
```

- List files and subdirectories in a directory

```
from oss.oss_api import *
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Listing files and subdirectories in the bucket's "fun/" directory
res= oss.list_bucket(“bucketname”, prefix=”fun/”, delimiter=”/”)
objects_xml=oss_xml_handler.GetBucketXml(res.read())
#Formatting output results
print "Objects:"
for object_info in objects_xml.content_list:
    print object_info.key

print "CommonPrefixs:"
for prefix in objects_xml.prefix_list:
    print prefix
```

```
Objects:
fun/test.jpg

CommonPrefixs:
fun/movie/
```

In the returned results, the Objects list contains the files in the fun directory.The CommonPrefixs list shows all subfolders in the fun directory. Obviously, the files fun/movie/001.avi and fun/movie/007.avi are not listed, because they are in the movie directory under the fun folder.

Retrieving Objects

Reading Objects

```
from oss.oss_api import *
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Reading the file
res=oss.get_object("bucketname","object")
print "%s\n%s\n%s " % (res.status, res.read(),res.getheaders())
```

The object content is returned in the HTTP response body and can be obtained using the res.read() method. Object header information can be obtained through res.getheaders(). It contains the ETag, Http Header, and custom metadata. At the same time, you can add the following HTTP headers in the request header for more detailed operations.

Parameter	Description
Range	Specifies the range of file transfer.
If-Modified-Since	If the specified time is earlier than the actual modification time, the file is transmitted normally. Otherwise, the system throws the 304 Not Modified exception.
If-Unmodified-Since	If the specified time is the same as or later than the actual modification time, the file is transmitted normally.Otherwise, the system throws the 412 precondition failed exception
If-Match	Imports an ETag group. If the imported expected ETag matches the object's ETag, the file is transmitted normally.Otherwise, the system throws the 412 precondition failed exception
If-None-Match	Imports an ETag group. If the imported ETag does not match the object's ETag, the file is transmitted normally.Otherwise, the system throws the 304 Not Modified exception.

```
from oss.oss_api import *
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Setting the request header, and reading the content from 20-100 bytes
headers={"Range":"bytes=20-100"}
res=oss.get_object("bucketname","object",headers)
print "%s\n%s\n%s" % (res.status, res.read(),res.getheaders())
```

We can set Range to return the object range. We can use this function for segmented file multipart download and resumable data transfer.

Directly Downloading Objects to Files

We can use the code below to directly download objects to a specified file:

```
from oss.oss_api import *
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Downloading the object to "file_path"
res=oss.get_object_to_file("bucketname","object",” file_path” )
print "%s\n%s\n%s" % (res.status, res.read(),res.getheaders())
```

Only Retrieve ObjectMetadata

```
from oss.oss_api import *
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Getting the object header information
res=oss.head_object("bucketname","object")
print "%s\n%s" % (res.status,res.getheaders())
```

Using head_object, we can retrieve the object header information, including the objectmeta.

Deleting Objects

```
from oss.oss_api import *
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Deleting objects
res=oss.delete_object("bucketname","object")
print "%s\n%s" % (res.status,res.read())
```

```
from oss.oss_api import *
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Batch deleting 3 objects
objectlist=[ “object1” ,” object2” ,” object3” ,]
res=oss.batch_delete_objects ("bucketname", objectlist)
print "Is success?"
print res
```

batch_delete_objects() returns a bool value, indicating if deletion was successful or not.

Copying Objects

You can copy an object with operation permissions within the same region.

Copying One Object

Using the copyObject method, we can copy a single object. The code is as follows:

```
from oss.oss_api import *
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Copying objects
res=oss. copy_object( “source_bucket” ,” source_object” ,” target_bucket” ,” target_object” ):
print "%s\n%s" % (res.status,res.getheaders())
```

Note that the source and destination buckets must be in the same region.

At the same time, the user is allowed to modify the ObjectMeta of the destination object and can use the following headers for more detailed operations.

Request Header	Description
x-oss-copy-source-if-match	If the source object's ETAG value is the same as the ETAG provided by the user, a copy operation will be executed. Otherwise, the system returns the 412 HTTP error code (preprocessing failed) Default value:none
x-oss-copy-source-if-none-match	If the source object has not been modified after the time specified by the user, the system performs a copy operation. Otherwise, the system returns the 412 HTTP error code (preprocessing failed) Default value:none
x-oss-copy-source-if-unmodified-since	If the time specified by the received parameter is the same as or later than the modification time of the file, the system transfers the file normally, and returns the 200 OK message; otherwise, the system returns the 412 Precondition Failed message. Default value:none
x-oss-copy-source-if-modified-since	If the source object has been modified after the time specified by the user, the system performs a copy operation. Otherwise, the system returns the 412 HTTP error code (preprocessing failed) Default value:none
x-oss-metadata-directive	If the parameter is set to COPY, the new object's meta is all copied from the source

	object. If the parameter is set to REPLACE, the source object's meta is ignored, and the meta values specified by the user in this request are used. Other values will cause the system to return an 400 HTTP error code. Note that when the value is COPY, the source object's x-oss-server-side-encryption meta value cannot be copied. Default value: COPY Valid values:COPY, REPLACE

```
from oss.oss_api import *
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#If the source ETag and provided ETag match, the copy operation is executed
headers={ "x-oss-copy-source-if-match" :" 5B3C1A2E053D763E1B002CC607C5A0FE" }
res=oss. copy_object( "source_bucket" ," source_object" ," target_bucket" ," target_object" ,headers):
print "%s\n%s" % (res.status,res.getheaders())
```

Modifying Object Meta

Copying data can modify the meta information of an existing object. If the address of the copied source object is the same as the address of the destination object, the source object's meta information will be replaced.

```
from oss.oss_api import *
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Setting the object's Content-Type
headers={ "Content-Type" :" image/japeg" }
res=oss. copy_object( "bucketname" ," object" ," bucketname" ," object" ,headers)
print "%s\n%s" % (res.status,res.getheaders())
```

Multipart Upload

Besides using the putObject interface to upload files to OSS, the OSS also provides a Multipart Upload mode.You can apply the Multipart Upload mode in the following scenarios (but not limited to the following):

- Where breakpoint uploads are needed.
- Uploading an object larger than 100MB.
- In poor network conditions, when the connection with the OSS server is frequently broken.
- When, before uploading the file, you cannot determine its size.

Below, we will give a step-by-step introduction to Multipart Upload.

Step-By-Step Multipart Upload

Initialization

Initializes a single multipart upload task

```
from oss.oss_api import *
from oss import oss_xml_handler
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Initializing a single multipart upload task
res=oss.init_multi_upload("bucketname","mutipartobject")
body = res.read()
if res.status == 200:
    #Using GetInitUploadIdXml to parse the xml and get the upload id
    h = oss_xml_handler.GetInitUploadIdXml(body)
    upload_id = h.upload_id
else:
    err = ErrorXml(body)
    raise Exception("%s, %s" %(res.status, err.msg))
```

The upload id is saved in the returned HTTP response body in xml format. Use the provided parsing method to obtain the upload id. UploadId is the unique identifier of a multipart upload task. We will use this in subsequent operations.

Upload Part Local Upload

Next, we will multipart upload the local file. Let us assume that there is one file in the local path '/path/to/file.zip'. Because it is large, we want to multipart upload it to OSS.

```
from oss.oss_api import *
from oss import oss_util
from oss import oss_xml_handler

filename=”  /path/to/file.zip”
max_part_num=20
filename = oss_util.convert_utf8(filename)
#Splitting the file and saving the information for each piece in the part_msg_list
part_msg_list = oss_util.split_large_file(filename, object, max_part_num)
uploaded_part_map = oss_util.get_part_map(oss, bucket, object, upload_id)
for part in part_msg_list:
#Extracting information for each piece
    part_number = str(part[0])
    partsize = part[3]
    offset = part[4]
    res = oss.upload_part_from_file_given_pos(bucket, object, filename, offset, partsize, upload_id, part_number)
#Saving etag
    etag = res.getheader("etag")
    if etag:
        uploaded_part_map[part_number] = etag
```

The main idea of this program is to call the uploadPart method to upload each part. However, you must note the following:

- In the uploadPart method, all parts except the last one must be larger than 100KB.
- In order to ensure that the data transmitted over the network is free from errors, we strong recommend that the user include meta: content-md5 in the request when uploading parts. After the OSS receives data, it uses this MD5 value to verify the correctness of the uploaded data. If it is not consistent, OSS returns the InvalidDigest error code.
- The part number range is 1~10000. If the part number exceeds this range, the OSS will return the InvalidArgument error code.
- When each part is uploaded, it will take the stream to the location corresponding to the start of the next part.

Completing Multipart Uploads

```
# Converting the part_msg_list generated by oss_util.split_large_file() to xml format
part_msg_xml = oss_util .create_part_xml(part_msg_list)
res = oss.complete_upload(bucket,objectname, upload_id, part_msg_xml,)
```

Canceling Multipart Upload Tasks

```
from oss.oss_api import *
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Using upload id to cancel a multipart upload task
oss.cancel_upload(bucketname, objectname, upload_id)
```

When a Multipart Upload event is aborted, you cannot use this Upload ID to perform any operations and the uploaded parts of data will be deleted.

Getting All Multipart Upload Tasks in the Bucket

```
from oss.oss_api import *
from oss import oss_util
from oss import oss_xml_handler
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Retrieving all multipart upload task information
res=oss.get_all_multipart_uploads(bucket)
#Parsing the results
res_xml= oss_xml_handler. GetMultipartUploadsXml(res.read())
for upload in res_xml.content_list
    print   "---------------------------------------"
    print   "key:"  +upload.key
    print   "uploaded:"  +upload.upload_id
```

All the upload event information is saved in the returned HTTP response body in xml format. Use the GetMultipartUploadsXml class to parse it.

Getting Information for All Uploaded Parts

```
from oss.oss_api import *
from oss import oss_util
from oss import oss_xml_handler
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Retrieving all part information for each multipart upload task

res=oss.get_all_parts(bucket,object,upload_id)
#Parsing the results
res_xml= oss_xml_handler. GetPartsXml (res.read())
for part in res_xml.content_list
    print   "---------------------------------------"
    print   "partnumber:"  +part. part_number
    print   "lastmodified:"  + part. last_modified
    print   "etag:"  + part. etag
    print   "size:"  + part. size
```

All the part information is saved in the returned HTTP response body in xml format. Use the GetPartsXml class to parse it.

Error Responses

Error Response Handling

If an error occurs when a user accesses the OSS, the OSS returns the error code and error message, so that the user can locate the problem and handle it properly. When the returned status is not 2XX, the error messages is saved in the response body in xml format. Refer to the processing of the following code:

```
from oss.oss_api import *
from oss import oss_xml_handler
oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Using the object list as an example
res= oss.list_bucket(bucket)
if res.status/100 !=  2:
    #Parsing the xml format error message
    err_msg=oss_xml_handler.ErrorXml(res.read())
    print 'Code:'+err_msg.code
    print 'Message:'+err_msg.msg
    print 'Request id:'+err_msg.request_id
else:
```

```
objects_xml=oss_xml_handler.GetBucketXml(res.read())
objects_xml.show()
```

Among them:

- Code: the error code OSS returns to users.
- Message: the detailed error message provided by OSS.
- RequestId: indicates a UUID of the request. If you cannot solve the problem, you can seek help from OSS development engineers by providing this RequestId.

Common Error Codes

Error Code	Description
AccessDenied	Access denied
BucketAlreadyExists	The bucket already exists
BucketNotEmpty	The bucket is not empty
EntityTooLarge	The entity is too large
EntityTooSmall	The entity is too small
FileGroupTooLarge	The file group is too large
FilePartNotExist	A file part does not exist
FilePartStale	A file part has expired
InvalidArgument	Parameter format error
InvalidAccessKeyId	The Access Key ID does not exist
InvalidBucketName	The bucket name is invalid
InvalidDigest	The digest is invalid
InvalidObjectName	The object name is invalid
InvalidPart	A part is invalid
InvalidPartOrder	The part order is invalid
InvalidTargetBucketForLogging	The logging operation has an invalid target bucket
InternalError	Internal OSS error
MalformedXML	Illegal XML format
MethodNotAllowed	The method is not supported
MissingArgument	A parameter is missing
MissingContentLength	The content length is missing
NoSuchBucket	The bucket does not exist
NoSuchKey	The file does not exist

NoSuchUpload	Multipart Upload ID does not exist
NotImplemented	The method cannot be processed
PreconditionFailed	Preprocessing error
RequestTimeTooSkewed	The request initiation time exceeds the server time by 15 minutes
RequestTimeout	Request timed out
SignatureDoesNotMatch	Signature error
TooManyBuckets	The user's bucket quantity exceeds the limit

Lifecycle Management

OSS provides the object lifecycle management capability to manage objects for users. The user can configure the lifecycle of a bucket to define various rules for the bucket's objects. Currently, users can use rules to delete matching objects. Each rule is composed of the following parts:

- The object name prefix; this rule will only apply to objects with the matched prefix.
- Operation; the operation the user wishes to perform on the matched objects.
- Date or number of days; the user will execute the operation on the objects on the specified date or a specified number of days after the object's last modification time.

Setting Lifecycles

The lifecycle configuration rules are expressed by an xml segment.

```
<LifecycleConfiguration>
  <Rule>
    <ID>delete obsoleted files</ID>
    <Prefix>obsoleted/</Prefix>
    <Status>Enabled</Status>
    <Expiration>
      <Days>3</Days>
    </Expiration>
  </Rule>
  <Rule>
    <ID>delete temporary files</ID>
    <Prefix>temporary/</Prefix>
    <Status>Enabled</Status>
    <Expiration>
      <Date>2022-10-12T00:00:00.000Z</Date>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

A single lifecycle Config can contain up to 1000 rules.

Explanations of each field:

- The ID field is used to uniquely identify a rule (inclusion relations, such as abc and abcd, cannot exist between IDs).
- Prefix indicates the rules used for objects in the bucket with the specified prefix.
- Status indicates the status of this rule. The statuses are Enabled and Disabled, indicating if the rule is enabled or disabled.
- In the Expiration node, Days indicates that an object will be deleted a specified number of days after its last modification. Date indicates that objects will be deleted after the specified absolute time (the absolute time follows the ISO8601 format).

Using the put_lifecycle() method, we can set a lifecycle rule for a specified bucket. If an original rule exists, it will be overwritten. Parameters for specific rules are generally set by uploading xml format strings. The code is as follows:

```
from oss.oss_api import *

oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Importing an xml string that sets lifecycle rules
res=oss.put_lifecycle(bucket,lifecycle_xml)
print res.status()
```

For setting lifecycle rules with xml format, refer to the API Documentation.

Retrieving Lifecycle Rules

We can refer to a bucket's lifecycle rules using the get_ lifecycle () method to retrieve them. The code is as follows:

```
from oss.oss_api import *

oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Retrieving CORS rule information
res=oss.get_cors(bucket)
#Saving the information in the response body in xml format
print res.read()
```

The read lifecycle rule information is saved in the response body in xml format and read by the read() method.

Deleting Lifecycle Rules

Use this to disable lifecycle for the specified bucket and clear all rules.

```
from oss.oss_api import *

oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Clearing the lifecycle rules set for the bucket
res=oss.delete_lifecycle (bucket)
print res.status()
```

Cross-Origin Resource Sharing (CORS)

CORS allows web applications to access resources in other regions. OSS provides an interface to allow developers to easily control cross-origin access permissions.

Setting CORS Rules

Using the put_cors() method, we can set a CORS rule for a specified bucket. If an original rule exists, it will be overwritten. Parameters for specific rules are generally set by uploading xml format strings. The code is as follows:

```
from oss.oss_api import *

oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Importing an xml string that sets a CORS rule
res=oss.put_cors(bucket,cors_xml)
print res.status()
```

For setting CORS in xml format, refer to OSS API Documentation.

Retrieving CORS Rules

We can refer to the bucket's CORS rules through the get_cors() method. The code is as follows:

```
from oss.oss_api import *
from oss import oss_xml_handler

oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Retrieving CORS rule information
res=oss.get_cors(bucket)
It is parsed using the provided CorsXml class
cors_xml=oss_xml_handler.CorsXml(res.read())
for rule in cors_xml.rule_list:
    print "----------------"
    rule.show()
```

The read CORS rule information is saved in the response body in xml format. Read the information with the read() method and use the CorsXml class for parsing.

Deleting CORS Rules

```
from oss.oss_api import *

oss = OssAPI(endpoint, accessKeyId, accessKeySecret)
#Clearing the CORS rules set for the bucket
res=oss.delete_cors(bucket)
print res.status()
```

Use this to disable CORS for the specified bucket and clear all rules.

Android-SDK

Preface

Description

This document introduces the methods for using the OSS Android SDK.

The OSS Android SDK is provided to help mobile developers more easily use the OSS cloud storage service on Android terminals. It is a storage component implemented based on OSS RESTful interfaces (AliCloud object storage service). By using this SDK, apps developed by developers can directly perform data access, deletion, copy, and other operations on the OSS server from a terminal. These operations provide two use modes: synchronous and asynchronous.

For more information on the API and some class structures, see the files in the SDK package doc/ directory.

You must note one thing: this SDK is a basic wireless terminal component developed based on OSS. It aims to satisfy mobile terminal developers' data storage needs. **It provides capabilities that allow mobile terminal applications to more easily access OSS data. It does not provide OSS Console management functions**, e.g., bucket application, bucket management, domain name binding, and subscription to static website hosting.

About OSS

Object Storage Service (OSS) is a cloud storage service provided by Alibaba Cloud, featuring massive capacity, security, and high reliability. This SDK provides Android mobile developers a set of OSS API interfaces tailor-made for the Android platform. Therefore, before using this SDK, you must first subscribe to the OSS service on the Alibaba Cloud official site and learn the basics of using OSS.

OSS Manual: OSS API Manual

SDK download

Android SDK 2.3.0: aliyun_OSS_Android_SDK_20160915

GitHub: click here

Sample: click here

Javadoc: click here

Installation

1. Use in the OneSDK

Currently, OSS mobile SDKs have already become a standard component of Alibaba Cloud OneSDK. You can use the OSS service by integrating OneSDK in an application. After integration, you just have to follow the OneSDK installation instructions. You do not have to configure any additional settings here, but can simply follow the steps in the following chapters to use the OSS service.

2. Direct Use in Eclipse

After downloading the OSS Android SDK zip package, follow the process below:

- Download the OSS Android SDK from the official website
- Decompress it and obtain the jar package
- Copy the jar package to the lib directory in your Android project
- In Eclipse, right-click Project -> Properties -> Java Build Path -> Add JARs to import the jar package you just copied

3. Permission Setting

The following are the Android permissions needed by the OSS Android SDK. Please make sure these

permission are already set in your AndroidManifest.xml file. Otherwise, the SDK will not work normally.

```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"></uses-permission>
<uses-permission android:name="android.permission.READ_PHONE_STATE"></uses-permission>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"></uses-permission>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"></uses-permission>
```

After completing the steps above, you can use the OSS Android SDK in the project.

Initialization

Before using the SDK, you must get the OSS Service in the SDK and then initialize some settings, e.g., implement the signature method, transfer the program Context, set the domain name of the default data center, etc.

In the whole application lifecycle, you only have to initialize these items once before using the OSS Android SDK.

1. Getting the OSS Service

The OSS Android SDK uses service IDs to provide various functions. The retrieval method is as follows:

```
OSSService ossService = OSSServiceProvider.getService();
```

In the application’ s lifecycle, you can get this ossService to use the OSS service multiple times.

2. Application Context Import

When the OSS Android SDK is working, it must use the application Context. Therefore, you must set this during initialization. This action is completed through the OSSService interface. The code is as follows:

```
ossService.setApplicationContext(getApplicationContext());
```

It must be pointed out that this operation is only effective the first time. Subsequent attempts to reset it will be ignored.

3. Setting the Data Center Domain Name

When creating a bucket on the OSS official site, you can choose a data center based on the unit price, the distribution of request sources, and the response latency. When you create a bucket, if you do not specify its data center, OSS will automatically allocate it a default data center. The current default data center is oss-cn-hangzhou.

Therefore, when performing OSS data operations, you must use a domain name to specify the data center of your bucket. This can be done through the following interface:

```
ossService.setGlobalDefaultHostId("oss-cn-qingdao.aliyuncs.com"); // Specifies that your bucket is placed in the Qingdao data center
```

If you **have not** called this interface to set the domain name, the OSS Android SDK will set your hostId to oss-cn-hangzhou.aliyuncs.com by default.

4. Token Generator Settings

For detailed information concerning token generator settings, refer to the [Access Control] chapter in this document. For now, you only need to know that, during initialization, you must perform this operation.

```
ossService.setAuthenticationType(AuthenticationType.ORIGIN_AKSK);
ossService.setGlobalDefaultTokenGenerator(new TokenGenerator() {
// Requires specific implementation
});
```

5. Custom Reference Time Setting

Because OSS token verification is time-sensitive, you may worry that incorrect system times of mobile terminals may cause users to be unable to access the OSS service. We have prepared an interface that allows you to set the SDK time. Over the network, you can get the current epoch time from the business server and set it. Then, during SDK operations, the time will be synced with the server time:

```
ossService.setCustomStandardTimeWithEpochSec(int currentEpochTimeInSec); // The epoch time is counted in seconds from January 1, 1970 00:00:00 UTC
```

Note that the time uses units of seconds.

6. Network-related Settings

The operations the OSS Android SDK performs on OSS resources all are completed through network

requests. Therefore, it is very likely that you will need to set the relevant network parameters. This can be done using a ClientConfiguration, e.g.:

```
ClientConfiguration conf = new ClientConfiguration();
conf.setConnectTimeout(15 * 1000); // Sets the connection establishment timeout time, default: 30s
conf.setSocketTimeout(15 * 1000); // Sets the socket timeout time, default: 30s
conf.setMaxConnections(50); // Sets the global maximum number of concurrent connections, default: 50
conf.setMaxConcurrentTaskNum(10); // Sets the global maximum number of concurrent tasks, default: 10
ossService.setClientConfiguration(conf);
```

7. Initialization Overview

During initialization, if you configure these settings, the initialization code section may be as follows:

```
OSSService ossService = OSSServiceProvider.getService();

ossService.setApplicationContext(getApplicationContext());
ossService.setGlobalDefaultHostId("oss-cn-hangzhou.aliyuncs.com");
ossService.setAuthenticationType(AuthenticationType.ORIGIN_AKSK);
ossService.setGlobalDefaultTokenGenerator(new TokenGenerator() {
// Requires specific implementation
});
ossService.setCustomStandardTimeWithEpochSec(currentEpochTimeInSec); // The epoch time is counted in seconds
from January 1, 1970 00:00:00 UTC

ClientConfiguration conf = new ClientConfiguration();
conf.setConnectTimeout(15 * 1000); // Sets the connection establishment timeout time, default: 30s
conf.setSocketTimeout(15 * 1000); // Sets the socket timeout time, default: 30s
conf.setMaxConnections(50); // Sets the global maximum number of concurrent connections, default: 50
conf.setMaxConcurrentTaskNum(10); // Sets the global maximum number of concurrent tasks, default: 10
ossService.setClientConfiguration(conf);
```

Access control

After subscribing to the OSS service on the Alibaba Cloud official site and creating your storage space (bucket), you can use the OSS Android SDK to access data on a terminal. To ensure the security of your data, OSS performs the appropriate security settings on the server. Accordingly, you need to implement the corresponding authentication process so that you can smoothly access your data.

Currently, the OSS Android SDK provides two authentication methods: the original AK/SK authentication and STS Token authentication. **STS Token authentication is better suited to mobile scenario authentication and it is the method we recommend**.

1. Original AK/SK authentication

When you register OSS, the system will assign you an Access Key ID and Access Key Secret pair, which is called the ID pair and referred to in this document as ‘AK/SK’. This is used for signature verification when you access OSS. For information on AK/SK usage, refer to User Signature Authentication. Put simply, each request you initiate to OSS must carry the token obtained from ‘AK/SK’ signing the request content. After successful authentication, the OSS server will process your request.

Because you are performing development on the terminals, the issues concerning secure storage of the AK/SK are hard to ignore. Therefore, the OSS Android SDK will not ask for your ‘AK/SK’. Rather, determined by your signature method, the SDK will call your signature method to generate a token prior to initiating an OSS request.

When using this method, you must call the following interface to set the authentication type:

```
ossService.setAuthenticationType(AuthenticationType.ORIGIN_AKSK);
```

Then, implement your token generator and set it in the ossService:

```
ossService.setGlobalDefaultTokenGenerator(TokenGenerator tokenGen);
```

TokenGenerator is an abstract object that contains an abstract method that you must implement. Its parameters correspond exactly to the Authorization field calculation method in the OSS API Manual. You just need to generate and return a Token by signing these parameters with AK/SK.

```
public abstract String generateToken(
String httpMethod,
String md5,
String type,
String date,
String ossHeaders,
String resource);
```

If we do not consider the security of the ‘AK/SK’, the sample code is as follows:

```
static final String accessKey = "Your accessKey"; // In actual use, the AK/SK is not stored in the code using clear text
static final String secretKey = "Your secretKey";
ossService.setGlobalDefaultTokenGenerator(new TokenGenerator() {
@Override
public String generateToken(String httpMethod, String md5, String type, String date, String ossHeaders,
String resource) {
String content = httpMethod + "\n" + md5 + "\n" + type + "\n" + date
+ "\n" + ossHeaders + resource;

return OSSToolKit.generateToken(accessKey, secretKey, content);
}
});
```

The tool functions used in the above code are already stored in the SDK, allowing you to use them for

testing. For your reference, the 'OSSToolKit.generateToken(...)' implementation details are as follows:

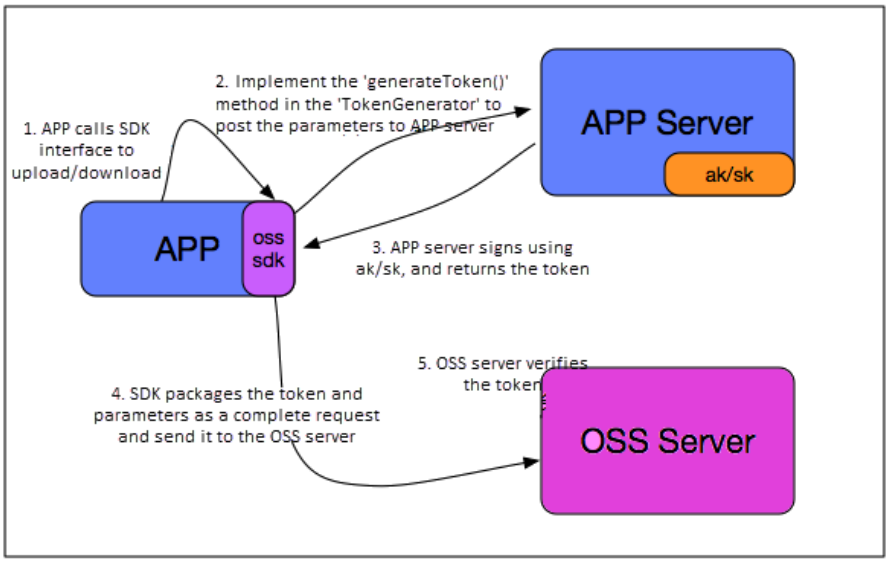
```
public static String generateToken(String accessKey, String secretKey, String content) {
String signature = null;

try {
signature = OSSToolKit.getHmacSha1Signature(content, secretKey);
signature = signature.trim();
} catch (Exception e) {
OSSLog.logD(e.toString());
}

return "OSS " + accessKey + ":" + signature; // Note there is a space between "OSS" and accessKey! ! !
}
```

It must again be emphasized that the 'AK/SK' should not be stored in the code using clear text. You should design a secure storage method for receiving the 'AK/SK'.

A suggested model is shown below:



As shown in the figure, you can store the 'AK/SK' on your business server, and then implement the 'generateToken()' method in the 'TokenGenerator' to post these parameters to your business server using an HTTP request. When your business server signs and returns the token, you then return this token to the 'TokenGenerator'. **This method is only called during actual uploads, downloads, and other operations**. Therefore, you do not have to worry about implantation involving network operations initiating the NetworkInMainThread exception. You just have to use a synchronized HTTP request to post the relevant fields to your business server and receive the token.

Therefore, in the entire process, the 'AK/SK' will not be exposed on the terminal, effectively

avoiding the risk of leaks.

It is difficult to combine security, ease-of-use, and efficiency. Before deciding on an AK/SK storage solution, please carefully weigh the risks and benefits.

2. STS token authentication

As described above, following the recommended security model and adopting the original AK/SK signature requires the terminal to get a signature result from the business server each time it initiates a request to the OSS service. When the terminal needs to access data frequently, the cost of this method will be quite high because a network interaction occurs each time the terminal gets a signature result.

In order to allow mobile terminals to conveniently and securely use the OSS service, Alibaba Cloud developed the STS service to support the security requirements in mobile scenarios. For details, refer to STS API Reference.

The OSS download package already contains STS Client SDKs for different programming languages, allowing the user to conveniently call the STS service on the business server. For the relevant use methods, refer to the demo in the package.

At the same time, we have provided a simple Android online storage demo based on STS. The source code is hosted on github and provided for reference: sts-demo-simple-android-app

In STS, Security Tokens are a core concept. They represent one-time authorizations you, as an ISV, can grant to terminal users. In fact, this authorization requires your original AK/SK (see Section 1 in this chapter). This authorization specifies the permission range and validity period. In the validity period, the Security Token can be used to directly sign requests in the permission range. This validity period generally covers one lifecycle of the mobile application, completely solving the problem described above. When required, it can also be updated.

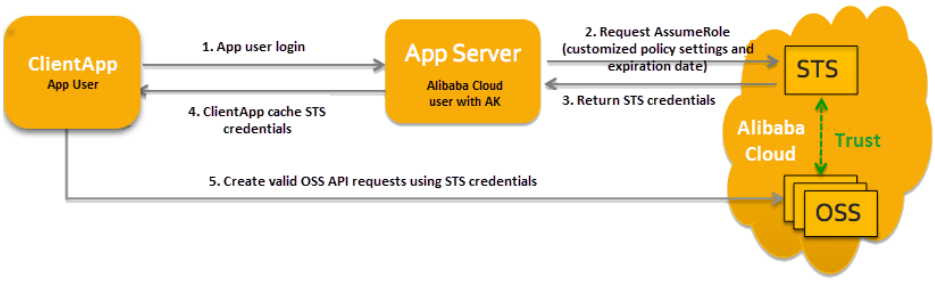
The use of the original AK/SK to get the Security Token and the OSS Android SDK behaviors are mutually independent. The SDK provides an interface for accepting the token, but it is not associated with your actual retrieval of the token. You must call the following interface:

```
ossService.setAuthenticationType(AuthenticationType.FEDERATION_TOKEN); // Sets the signature method as STS Token authentication
ossService.setGlobalDefaultStsTokenGetter(StsTokenGetter stsTokenGetter);
```

First set the ossService authentication method as Token. Then, set the STS token generator in the ossService. The implementation of this token generator depends on your STS Token retrieval method. This is the same as AK/SK authentication. The difference is that the STS Security Token can be used throughout the validity period. Therefore, in the validity period, the SDK will continue to use this token internally until it expires. Then, it will call this interface again to get another one. However, when you call the 'setGlobalDefaultStsTokenGetter()' interface again, the cached token will be forced to expire, allowing you to switch authorizations conveniently.

```
ossService.setAuthenticationType(AuthenticationType.FEDERATION_TOKEN); // Sets the signature method as STS FederToken authentication
ossService.setGlobalDefaultStsTokenGetter(new StsTokenGetter() {
// Write your code for getting a new STS Token here
// Under normal conditions, this token should be obtained through a network request to your business server
// Note that the returned OSSFederationToken must include four valid fields:tempAK/tempSK/securityToken/expiration
// The expiration value is the token's expiration time, format: UNIX Epoch time. This is the number of seconds from January 1, 1970 00:00:00 in coordinated universal time
});
```

A suggested model is shown below:



As shown in the figure, the STS service provides servers with SDKs for multiple programming languages. This allows you to conveniently get the STS Token from your business server and grant it to mobile terminals.

Bucket Operations

1. Bucket settings

Buckets are the namespaces in OSS. They are also the management entities for billing, permission control, logging, and other advanced functions. Naturally, prior to performing data-related operations, you must first create a bucket instance and configure it. This lets you easily use this bucket instance to specify data storage locations in subsequent data operations.

Creating a bucket is very simple:

```
OSSBucket sampleBucket = ossService.getOssBucket("oss-example");
```

Then, you can configure some optional settings.

1.1 Stating Bucket Access Permissions

When creating a bucket on the OSS official site, you can specify three access permissions: PUBLIC_READ_WRITE, PUBLIC_READ, and PRIVATE. Therefore, in SDK, you also need to specify the permissions set for the bucket you wish to access. This statement will not change your console settings.

```
sampleBucket.setBucketACL(AccessControlList.PRIVATE); // States the access permission for this bucket
```

1.2 Setting the Data Center Domain Name or CNAME

This can simply specify the bucket's data center or sets the domain name already bound to the bucket:

```
sampleBucket.setBucketHostId("oss-cn-hangzhou.aliyuncs.com"); // Specifies the domain name of the bucket's data center or the CNAME domain name already bound to the bucket
```

1.3 Setting CDN Domains

Buckets can simply set the CNAME domain name of the CDN domain they direct to. Because CDN does not currently support nearest upload, **if you wish to set this, it can only be used during download**. For example:

```
sampleBucket.setCdnAccelerateHostId("cname.to.cdn.domain.com"); // Sets the CNAME domain name for the CDN domain directed to
```

2. Initialization Overview

When initializing a bucket, if you configure all settings, the initialization code section may be as follows:

```
OSSBucket sampleBucket = ossService.getOssBucket("oss-example");
sampleBucket.setBucketACL(AccessControlList.PRIVATE); // Indicates the access permission for this bucket
sampleBucket.setBucketHostId("oss-cn-hangzhou.aliyuncs.com"); // Specifies the domain name of the bucket's data center or the CNAME domain name already bound to the bucket
// sampleBucket.setCdnAccelerateHostId("cname.to.cdn.domain.com"); // Sets the CNAME domain name for the CDN domain directed to
```

Normally, buckets can be regarded as a global concept because, in most situations, all data operations are associated with a bucket. Therefore, you should create these buckets during global initialization. During subsequent data operations, you only need to import the corresponding bucket to the specified data location.

3. List Objects

After setting a bucket, you can call the following interface to list the objects in the bucket:

```
public ListObjectResult listObjectsInBucket(ListObjectOption opt);
```

ListObjectOption is an optional parameter, used to set the Max-Keys, Marker, Prefix, and Delimiter parameters. The significance of these parameters is consistent with the descriptions in **OSS API Manual**. ListObjectResult is the returned result from which you can extract the relevant information. For example:

```
ListObjectOption opt = new ListObjectOption();
opt.setDelimiter("/");
opt.setPrefix("prefixdir/");
opt.setMaxKeys(500);
opt.setMarker( "prefix187");

ListObjectResult result = sampleBucket.listObjectsInBucket(opt); // Calls the interface to list bucket objects

int objectNum = result.getObjectInfoList().size();
String nextMarkder = result.getNextMarker();
Boolean isTruncated = result.isTruncated();
List<ObjectInfo> objectList = result.getObjectInfoList(); // Retrieves the objects list from the returned results
List<String> commonPrefixes = result.getCommonPrefixList(); // When Delimiter is set, objects with matching
prefixes will be grouped together and recorded in the returned commonprefix

for (ObjectInfo ele : objectList) { // Traverses the returned objects list
    String objectKey = ele.getObjectKey();
    String lastModifiedTime = ele.getLastModified());
}

for (String prefix : commonPrefixes) {
    // doSomething with prefixes
}
```

Note: This method is a sync method. During execution, it will send network requests, so do not directly call in the main thread.

Data Storage

The OSS Android SDK performs data storage operations using 'OSSData'. Here, data indicates the data in the memory when the application is running. Therefore, if you need to upload data in the memory when the application is running or wish to download a line of data from OSS to the local memory as a byte[] array to process it, you should perform an operation of this type.

When constructing an 'OSSData' instance, you must specify 'OSSBucket' and 'ObjectKey'. This

indicates the data line in OSS to be operated on. Instance retrieval is as follows:

```
OSSData ossData = ossService.getOssData(OSSBucket ossBucket, String ObjectKey);
```

1. Data Downloads

After you construct an 'OSSData' instance, if the 'OSSBucket' and 'ObjectKey' you specify correspond to data existing in OSS, you can download it. The code is as follows:

```
OSSData ossData = ossService.getOssData(sampleBucket, "sample-data"); // Constructs an OSSData instance
byte[] data = ossData.get(); // A failed download will throw an exception
```

Here, you must note the following:

- The above is just the simplified sample code. For actual applications, the download may fail for various reasons, so you should write code to catch exceptions and then handle them. For exception explanations, refer to the following chapter.
- This is a synchronous method and may clog the thread until the method is returned.
- When performing data operations, the OSS Android SDK must perform network interactions with the OSS server. Therefore, this code is considered a time-consuming obstructive operation. On Android platforms, most codes are run on the main thread (also called the UI thread). If the main thread has a time-consuming, obstructive operation, your codes may not run normally. Therefore, you should execute these codes on a non-UI thread. Or, you can use the asynchronous 'getInBackgroud()' interface provided by the SDK. For details, please refer to section 2 in this chapter.

In addition, **if your file is in a folder**, you just need to write the path in 'ObjectKey':

```
OSSData ossData = ossService.getOssData(sampleBucket, "fileFolder/sample-data"); // Constructs an OSSData instance
byte[] data = ossData.get(); // A failed download will throw an exception
```

After the data download is completed, you can call the 'getMeta()' method to retrieve the metadata:

```
List<BasicNameValuePair> metaList = OSSData.getMeta();
```

This method can only get results after the data download is completed (if it an asynchronous download, after 'onSuccess()' is triggered). If you wish to directly retrieve metadata, please refer to section 8.

2. Asynchronous Data Download Version

In fact, the OSS Android SDK provides an asynchronous version for all time-consuming methods,

allowing you to import your own implemented callback method. This performs the operation in a new thread and asynchronously calls back your processing logic.

For data downloads, the asynchronous interface is used as follows:

```
OSSData ossData = ossService.getOssData(sampleBucket, "sample-data");
ossData.getInBackground(new GetBytesCallback() {
    @Override
    public void onSuccess(String objectKey, byte[] data) {
        // Here, the downloaded data is processed
    }

    @Override
    public void onProgress(String objectKey, int byteWirtten, int totalSize) {
        // Here, you can perform operations based on the download progress
    }

    @Override
    public void onFailure(String objectKey, OSSException ossException) {
        // When a download fails, you can get the exception information and perform exception handling here
    }
});
```

When using the asynchronous interface, please **be sure to import a callback method**. If you do not need a callback method, you can leave it blank.

It must be pointed out that, when you implement these callback methods, they will be called by the OSS Android SDK in the task processing thread. This is to say that they will be executed in a non-UI thread. In Android, non-UI threads cannot control UI elements. Therefore, if you need to control a UI element in a callback method, you must do this through the message passing mechanism provided by Android. There are already a good deal of reference materials for this process, so we will not describe it again here.

The reason we do not first transfer the callback method to the UI thread for execution in SDK is that, if this is done with all callback methods, it will severely restrict the flexibility of SDK usage scenarios.

3. Retrieving the Data Input Stream

When downloading data, you can also directly retrieve the data input stream to perform the necessary processing:

```
OSSData ossData = ossService.getOssData(sampleBucket, "sample-data"); // Constructs an OSSData instance
try {
    InputStream inputStream = ossData.getObjectInputStream(); // A failed retrieval will throw an exception
    // do something with inputStream
} catch (OSSException ossException) {

}
```

4. Data Uploads

If, when running an application, you must upload the data in the memory to OSS, you must first specify the 'OSSBucket' and 'ObjectKey' to construct an 'OSSData' instance. Then, call the 'setData' interface and specify the data to upload and its type. Finally, call the upload method to upload the data.

When uploading, you can select whether or not the MD5 check is enabled.

The code is as follows:

```
OSSData ossData = ossService.getOssData(sampleBucket, "sample-data");
ossData.setData(data, "raw"); // Specifies the data to upload and its type
ossData.enableUploadCheckMd5sum(); // Enables the upload MD5 check
ossData.upload(); // Failed uploads will throw an exception
```

After the upload is completed, the bucket corresponding to 'sampleBucket' on OSS will have a new line of data, named 'sampleData'. Its content is the 'data' you uploaded.

Likewise, data upload also has an asynchronous version:

```
OSSData ossData = ossService.getOssData(sampleBucket, "sample-data");
ossData.setData(data, "raw"); // Specifies the data to upload and its type
ossData.enableUploadCheckMd5sum(); // Enables the upload MD5 check
ossData.uploadInBackground(new SaveCallback() {
    @Override
    public void onSuccess(String objectKey) {

    }

    @Override
    public void onProgress(String objectKey, int byteCount, int totalSize) {

    }

    @Override
    public void onFailure(String objectKey, OSSException ossException) {

    }
});
```

You can set a data stream as the data to be uploaded. However, when you do so, you must specify the length of the upload data to read from the data stream. For example:

```
OSSData ossData = ossService.getOssData(sampleBucket, "sample-data");

File fileToUpload = new File("<path/to/file>");
InputStream in = new FileInputStream(fileToUpload);

ossData.setInputstream(in, (int) fileToUpload.length(), "raw");
```

```
ossData.enableUploadCheckMd5sum(); // Enables the upload MD5 check

// ossData.upload(); // Failed synchronous uploads will throw an exception

ossData.uploadInBackground(new SaveCallback() {
    @Override
    public void onSuccess(String objectKey) {

    }

    @Override
    public void onProgress(String objectKey, int byteCount, int totalSize) {

    }

    @Override
    public void onFailure(String objectKey, OSSException ossException) {

    }
});
```

5. Data Deletion

After you construct an 'OSSData' instance representing the data corresponding to the specified 'OSSBucket' and 'ObjectKey', you can call the delete method to delete this line of data. The code is as follows:

```
OSSData ossData = ossService.getOssData(sampleBucket, "sample-data"); // Constructs an OSSData instance
ossData.delete(); // Failed deletions will throw an exception
```

Likewise, there is an asynchronous version:

```
OSSData ossData = ossService.getOssData(sampleBucket, "sample-data"); // Constructs an OSSData instance
data.deleteInBackground(new DeleteCallback() {
    @Override
    public void onSuccess(String objectKey) {

    }

    @Override
    public void onFailure(String objectKey, OSSException ossException) {

    }
});
```

The delete operation has no progress status. You do not need a callback method for the implementation progress. The copy method below is similar.

6. Data Copying

You can construct an 'OSSData' instance not existing in OSS for the specified 'OSSBucket' and 'ObjectKey'. Then, call the copy method to copy other existing OSS data to it. The copy source can be data in the same or a different bucket.

The code is as follows:

```
OSSData ossData = ossService.getOssData(sampleBucket, "sample-data");
ossData.copyFrom("sourceData"); // Copies from the same bucket
```

//Or:

```
ossData.copyFrom("sourceBucket", "sourceData"); // Copies from a different bucket
```

Asynchronous version:

```
OSSData ossData = ossService.getOssData(sampleBucket, "sample-data");
ossData.copyFromInBackgroud("sourceData", new CopyCallback() {
    @Override
    public void onSuccess(String objectKey) {

    }

    @Override
    public void onFailure(String objectKey, OSSException ossException) {

    }
});
```

//Or:

```
ossData.copyFromInBackgroud("sourceBucket", "sourceData", new CopyCallback() {
    @Override
    public void onSuccess(String objectKey) {

    }

    @Override
    public void onFailure(String objectKey, OSSException ossException) {

    }
});
```

7. Specifying a Range During Downloads

Before downloading data, you can set 'Range' to specify the download range:

```
ossData.setRange(222, 888)); // This is only valid for a download operation and, to be effective, must be called prior
```

```
to the download
ossData.setRange(222, Range.INFINITE); // Sets the download range as 222 to the end of the object
OSSData.setRange(Range.INFINITE, 20); // Indicates a download of the last 20 bytes of the object
```

8. Adding Custom Meta Attributes During Uploads

Data in OSS has its own meta attributes. Besides being automatically added by the system, you can also specify custom attributes using the "x-oss-meta-" prefix. For details, please refer to the OSS product documentation.

The attributes you specify at the time of upload can be individually retrieved through the retrieve meta attributes method in the SDK. For details, refer to the [Only Retrieve Meta] chapter in this document.

The code for adding custom meta attributes prior to upload is as follows:

```
ossData.addXOSSMetaHeader("x-oss-meta-key1", "value1"); // This is only valid for upload operations and, to be effective, must be called prior to the upload
ossData.addXOSSMetaHeader("x-oss-meta-key2", "value2"); // Custom meta attributes must have the "x-oss-meta-" prefix
ossData.addXOSSMetaHeader("x-oss-meta-key3", "value3"); // Meta attribute key-value pairs of the same name are not supported
```

Note: If you add meta attributes without the "x-oss-meta-" prefix, they will be ignored. Also, we do not suggest adding information in Chinese in meta. Otherwise, you must manually perform a codec operation. This means you must encode it before uploading, and manually decode it after downloading.

9. Generating Data URLs

After constructing the data for the specified 'OSSBucekt' and 'objectKey', you can call the 'getResourceURL()' interface to generate an access URL. This can be used to authorize URL access by third-parties. If the data's bucket does not have 'Public-Read' permission, you must import an 'accessKey' and a validity period for this interface. This URL will expire after the end of the validity period. If you are using the STS authentication model, this 'accessKey' can import any value.

```
OSSData.getResourceURL(String accessKey, int availablePeriodInSeconds); // The generated URL will expire in availablePeriodInSeconds seconds
```

If your bucket has the 'Public-Read' permission, you just have to call:

```
OSSData.getResourceURL(); // Because the Bucket has Public-Read permission, you do not need to import an accessKey or validity period
```

10. Canceling Asynchronous Upload/Download Tasks

For mobile terminals, and especially those that use a 2G/3G network, upload and download operations for large files may take a long time. In the middle of processing such a task, you may have to discard it for some reason. Therefore, the OSS Android SDK provides a cancel function for data asynchronous upload/download and file asynchronous upload/download interfaces for these scenarios. After starting an asynchronous task, the interface will return a 'TaskHandler', where you can call the 'cancel()' interface to cancel this task. If the task is successfully canceled, it will enter 'onFailure()' and throw an 'InterruptIOException' exception.

```
TaskHandler tHandler = OSSData.getInBackground(new GetBytesCallback() {...}); // Starts an asynchronous download task
// doSomething
tHandler.cancel(); // The task must be canceled for some reason
```

File Operations

In the OSS Android SDK, file operations are collected in 'OSSFile'. This is generally used in the same way as 'OSSData'. In fact, the **differences concern the fact that** 'OSSData' can be used to retrieve data in the memory, while 'OSSFile' is used to directly upload local files on the terminal and download OSS data to local files.

If the upload file is large and takes a long time, an exception may occur during the process, causing the upload to fail. Therefore, the OSS Android SDK provides a breakpoint resumption interface for large file uploads.

1. Downloading to Files

The file download operation is similar to the data download operation. This only difference is that, during the download, you must specify the destination file path. The download method will not return data.

```
OSSFile ossFile = ossService.getOssFile(sampleBucket, "sample-data");
ossFile.downloadTo("/path/to/file"); // Failed downloads will throw an exception
```

Asynchronous version:

```
ossFile.downloadToInBackground("/path/to/file", new GetFileCallback() {
    @Override
    public void onSuccess(String objectKey, String filePath) {

    }
```

```
    @Override
    public void onProgress(String objectKey, int byteCount, int totalSize) {

    }

    @Override
    public void onFailure(String objectKey, OSSException ossException) {

    }
});
```

As you can see, the asynchronous interface's 'onSuccess' callback method no longer has a byte[] array parameter. It is replaced by the file save path after successful download.

3. Retrieving a File Input Stream

Likewise, when downloading an object, you can also directly retrieve the input stream to perform the necessary processing:

```
OSSFile ossFile = ossService.getOssFile(sampleBucket, "sample-data"); // Constructs an OSSFile instance
try {
    InputStream inputStream = ossFile.getObjectInputStream(); // A failed retrieval will throw an exception
    // do something with inputStream
} catch (OSSException ossException) {

}
```

3. Uploading From Files

The upload from file interface has also changed. Before the upload, you must specify the upload file's path.

```
OSSFile ossFile = ossService.getOssFile(sampleBucket, "sample-data");
ossFile.setUploadFilePath("/path/to/file", "content type"); // Specifies the path of the file to be uploaded and the file content type. If the file does not exist, the system will throw an exception
ossFile.enableUploadCheckMd5sum(); // Enables the upload MD5 check
ossFile.upload(); // Failed uploads will throw an exception
```

Asynchronous version:

```
ossFile.uploadInBackground(new SaveCallback() {
    @Override
    public void onSuccess(String objectKey) {

    }
```

```
    @Override
    public void onProgress(String objectKey, int byteCount, int totalSize) {

    }

    @Override
    public void onFailure(String objectKey, OSSException ossException) {

    }
});
```

4. Other Common Operations

The two sections above have already described the differences between 'OSSFile' and 'OSSData'. As you can see, from the perspective of OSS, there is no difference between them, as they are both data segments stored in OSS. Therefore, except for the differences of the upload/download interfaces concerning the data format, **the usage of the other interfaces is the same. This includes copying, deleting, specifying a download range, adding custom meta attributes, generating URLs, and canceling jobs**.

Thus, we will not describe them here.

5. Breakpoint Downloads

Because large file downloads are time-consuming, there is a high risk that the process will fail. Therefore, the OSS Android SDK provides a breakpoint download interface. When you use this interface to download files, if the download fails midway through for some reason, you can specify the same 'OSSbucket' and 'ObjectKey' for the next download, without changing the file save path. Then, the next download will continue from where the previous download failed.

At the same time, when you call this interface, it will return a task 'handler'. During the download process, you can use this 'handler' to cancel the download task at any time. After the task fails, it will enter the 'onFailure()' logic.

This interface only has an asynchronous version. The sample code is as follows:

```
OSSFile ossFile = ossService.getOssFile(sampleBucket, "test.jpg");
TaskHandler tk = ossFile.ResumableDownloadToInBackground("/path/to/file", new GetFileCallback() {
    @Override
    public void onSuccess(String objectKey, String filePath) {
        // TODO Auto-generated method stub
    }

    @Override
    public void onProgress(String objectKey, int byteCount, int totalSize) {
        // TODO Auto-generated method stub
    }

    @Override
```

```
    public void onFailure(String objectKey, OSSException ossException) {
        // TODO Auto-generated method stub
    }
});

tk.cancel(); // During the download task execution, you can discard the download at any time
```

You can specify some settings for breakpoint downloads, including the recorded file storage path, maximum number of concurrent threads, and number of auto retries. If you do not set these options, they will take their default values.

```
OSSFile ossFile = ossService.getOssFile(sampleBucket, "test.jpg");

ResumableTaskOption option = new ResumableTaskOption();
option.setThreadNum(2); // Default: 3, maximum: 5
option.setRecordFileDirectory("record/file/dir"); // Stored in the application data directory by default
option.setAutoRetryTime(1); // Default: 2, maximum: 3

TaskHandler tk = ossFile.ResumableDownloadToInBackground("/path/to/file", option, new GetFileCallback() {
    @Override
    public void onSuccess(String objectKey, String filePath) {
        // TODO Auto-generated method stub
    }

    @Override
    public void onProgress(String objectKey, int byteCount, int totalSize) {
        // TODO Auto-generated method stub
    }

    @Override
    public void onFailure(String objectKey, OSSException ossException) {
        // TODO Auto-generated method stub
    }
});
```

6. Resuming From Breakpoints

Likewise, the OSS Android SDK also provides a breakpoint resumption interface. When you use this interface to upload files, if the upload fails midway through for some reason, you can specify the same 'OSSbucket' and 'ObjectKey' for the next upload, without changing the file to upload. Then, the next upload will continue from where the previous upload failed.

At the same time, when you call this interface, it will return a task 'handler'. During the upload process, you can use this 'handler' to cancel the upload task at any time. After the task fails, it will enter the 'onFailure()' logic.

This interface only has an asynchronous version. The sample code is as follows:

```
OSSFile ossFile = ossService.getOssFile(sampleBucket, "large.data");
ossFile.setUploadFilePath("/path/to/file", "file content type"); // Specifies the path of the file to be uploaded and the
```

```
file content type
TaskHandler tk = ossFile.ResumableUploadInBackground(new SaveCallback() {
    @Override
    public void onSuccess(String objectKey) {
        // TODO Auto-generated method stub
    }

    @Override
    public void onProgress(String objectKey, int byteCount, int totalSize) {
        // TODO Auto-generated method stub
    }

    @Override
    public void onFailure(String objectKey, OSSException ossException) {
        // TODO Auto-generated method stub
    }
});

tk.cancel(); // During the upload task execution, you can discard the upload at any time
```

Likewise, you can specify some settings for breakpoint resumption tasks. These settings are the same as for breakpoint downloads in the previous chapter.

```
OSSFile ossFile = ossService.getOssFile(sampleBucket, "test.jpg");

ResumableTaskOption option = new ResumableTaskOption();
option.setThreadNum(2); // Default: 3, maximum: 5
option.setRecordFileDirectory("record/file/dir"); // Stored in the application data directory by default
option.setAutoRetryTime(1); // Default: 2, maximum: 3

TaskHandler tk = ossFile.ResumableUploadInBackground(option, new SaveCallback() {
    @Override
    public void onSuccess(String objectKey) {
        // TODO Auto-generated method stub
    }

    @Override
    public void onProgress(String objectKey, int byteCount, int totalSize) {
        // TODO Auto-generated method stub
    }

    @Override
    public void onFailure(String objectKey, OSSException ossException) {
        // TODO Auto-generated method stub
    }
});
```

Retrieving Meta

You can use OSSMeta to independently retrieve data's meta attributes. The collection of retrieved meta attributes are returned to you in the List<BasicNameValuePair> format.

The sample code is as follows:

```
OSSMeta meta = ossService.getOssMeta(sampleBucket, "sample-data");
List<BasicNameValuePair> namevalue = meta.getMeta();
```

Asynchronous version:

```
OSSMeta meta = ossService.getOssMeta(sampleBucket, "sample-data");
meta.getMetaInBackground(new GetMetaCallback() {
    @Override
    public void onSuccess(String objectKey, List<BasicNameValuePair> meta) {

    }

    @Override
    public void onFailure(String objectKey, OSSException ossException) {

    }
});
```

Independent Multipart Uploads

If breakpoint uploads do not meet your needs, the SDK provides an independent multipart upload interface, allowing you to control your upload parts. There are five interfaces associated with multipart uploads: get UploadId, upload a single part, complete upload, list uploaded parts, and cancel upload. These interfaces are used through 'OSSMultipart'. When carrying out a multipart upload task, you can continue using a 'OSSMultipart' object, or call different 'OSSMultipart' objects when calling each interface. For example:

```
String asPartFile = "<path/to/file>"; // Splits the large file to be uploaded into two upload parts
String objectkey = "bigFile.dat";
OSSMultipart multipart = ossService.getOssMultipart(bucket, objectkey);
multipart.setContentType("binary/raw");

String uploadId = multipart.initiateMultiPartUpload();

File file = new File(asPartFile);
InputStream inputStream = new FileInputStream(file);
multipart.setUploadpart(1, inputStream, 128 * 1024); // The first part uploads the first 128kb
multipart.uploadPart();

byte[] data = ToolKit.readFullyToByteArray(inputStream);
multipart.setUploadpart(2, data); // The second part uploads the remaining data
multipart.uploadPart();

List<UploadPartInfo> info = multipart.listParts(); // If needed, this shows which parts have been uploaded
```

```
multipart.completeMultipartUpload();
```

As you can see, when setting the data to be uploaded for a single part, it can be read from the data stream or you can directly set a byte array as the data. In addition, if you need to use a different 'OSSMultipart' object for each stage to meet your special needs, you must save the information from the previous stage as the upload ID of the next stage. For example:

```
String asPartFile = "<path/to/file>"; // Splits the large file to be uploaded into two upload parts
String objectkey = "bigFile.dat";
OSSMultipart multipart1 = ossService.getOssMultipart(bucket, objectkey);
multipart1.setContentType("binary/raw");

uploadId = multipart1.initiateMultiPartUpload();

List<UploadPartResult> list = new ArrayList<>();

OSSMultipart multipart2 = ossService.getOssMultipart(bucket, objectkey);
multipart2.designateUploadId(uploadId); // The necessary upload ID, specifies the corresponding upload task

file = new File(asPartFile);
inputStream = new FileInputStream(file);
multipart2.setUploadpart(1, inputStream, 128 * 1024);
result1 = multipart2.uploadPart();
list.add(result1);

OSSMultipart multipart3 = ossService.getOssMultipart(bucket, objectkey);
multipart3.designateUploadId(uploadId); // The necessary upload ID, specifies the corresponding upload task

byte[] data = ToolKit.readFullyToByteArray(inputStream);
multipart3.setUploadpart(2, data);
result2 = multipart3.uploadPart();
list.add(result2);

// If needed, this shows which parts have been uploaded
OSSMultipart multipart4 = ossService.getOssMultipart(bucket, objectkey);
multipart4.designateUploadId(uploadId); // Specifies the corresponding upload task
List<UploadPartInfo> info = multipart4.listParts();

OSSMultipart multipart5 = ossService.getOssMultipart(bucket, objectkey);
multipart5.designateUploadId(uploadId); // Specifies the corresponding upload task
multipart5.designatePartList(list); // Specifies the information of the locally saved parts that have been uploaded
multipart5.completeMultipartUpload();
```

It must be emphasized that these interfaces are all time-consuming network call operations. They can clog the current thread and may throw an OSSException. For exception handling, refer to the exception handling chapter below

Complete Examples

For complete examples, please refer to the Demo project in the SDK package.

Exception Handling

The OSS Android SDK has many interfaces that must perform network interaction with the OSS server. Each request must be authenticated independently or involves operations on local files, etc. These actions may encounter exceptions. Therefore, SDK provides a specialized OSSException, responsible for collecting exception information and feedback.

All exceptions will be packaged in OSSException. When using synchronous interfaces, exceptions are thrown. When using asynchronous interfaces, exceptions will be transmitted to the callback method onFailure(String objectKey, OSSException ossException) which must be executed upon failure for processing.

Exceptions are classified as follows: local exceptions and OSS exceptions. The former indicates that the system encountered an exception, e.g., network connection failure, IO exception, file exception, parameter exception, or status exception. The later indicates that the OSS system could not process a request, e.g., non-existent resource, authentication failed, or an object too is large.

Before processing exceptions, you should determine the exception type.

1. Local Exceptions

If you have caught an ossException, you can retrieve the exception information and perform the relevant processing as follows:

```
try {
//...
} catch (OSSException ossException )  {
//Before processing, you must determine the exception type, otherwise the processing method may not be applicable to the type
if (ossException.getExceptionType() == ExceptionType.LOCAL_EXCEPTION) {
String objectKey = ossException.getObjectKey(); // Retrieves the ObjectKey for this task
String mesString = ossException.getMessage(); // The exception information
String info = ossException.toString();
Exception localException = ossException.getLocalException(); // Gets the original exception
localException.printStackTrace(); // Prints the stack
}
}
```

2. OSS Exceptions

If you have caught an ossException, you can retrieve the exception information and perform the relevant processing as follows:

```
try {
//...
} catch (OSSException ossException )  {
//Before processing, you must determine the exception type, otherwise the processing method may not be applicable to the type
if (ossException.getExceptionType() == ExceptionType.OSS_EXCEPTION) {
String objectKey = ossException.getObjectKey(); // Retrieves the ObjectKey for this task
OSSResponseInfo resp = ossException.getOssRespInfo(); // Retrieves the data structure based on the construction of the OSS response content
int statusCode = resp.getStatusCode(); // The OSS response's http status code
Document dom = resp.getResponseInfoDom(); // The file structure obtained by parsing the OSS response content. You can use this to get more detailed information
String errorCode = resp.getCode(); // The error code fed back by OSS
String requestId = resp.getRequestId(); // The request ID for this task
String hostId = resp.getHostId(); // The request host for this task
String message = resp.getMessage(); // The error message fed back by OSS
String info = ossException.toString(); // The information summary for this exception
ossException.printStackTrace(); // Prints the stack
}
}
```

This exception information is constructed and processed based on the standard error responses in the OSS API Documentation file.

iOS-SDK

Preface

Description

This document introduces the methods of using the OSS iOS SDK.

A storage component based on OSS RESTful interfaces (AliCloud object storage service), the OSS iOS SDK is provided to help mobile developers more easily use the OSS cloud storage service on iOS terminals. By using this SDK, apps developed by developers can directly perform data access, deletion, copy, and other operations on the OSS server from a terminal. These operations work in two modes: synchronous and asynchronous.

Please note that this SDK is a basic wireless terminal component developed based on OSS. It aims to satisfy mobile terminal developers' data storage needs. **It provides capabilities to facilitate the access to OSS data by mobile terminal applications. It does not provide OSS Console management functions**, e.g. bucket application, bucket management, virtual hosting, and activating static website

hosting. Nor does the SDK support global browsing of bucket data. The mapping relationships between data must be maintained by the developer.

About OSS

Object Storage Service (OSS) is a cloud storage service provided by AliCloud, featuring massive capacity, security, and high reliability. This SDK provides iOS mobile developers with a set of OSS API interfaces tailor-made for the iOS platform. Therefore, before using this SDK, you must activate the OSS service on the AliCloud official site and learn the basics about OSS usage.

OSS Manual: OSS API Manual

SDK download

- iOS SDK 2.5.2: aliyun_OSS_iOS_SDK_20160828.zip
- GitHub地址：https://github.com/aliyun/aliyun-oss-ios-sdk
- Pod dependency: pod 'AliyunOSSiOS', '~> 2.5.2'
- demo: https://github.com/alibaba/alicloud-ios-demo

Installation

We provide this SDK in Static Library format.

1. Direct Use in Xcode

Select Your Projects -> TARGETS -> Your Items -> General -> Linked Frameworks and Libraries -> Click “+” -> add other -> the directory containing the framework -> select the framework file -> open

2. Introducing Header Files

```
#import <ALBB_OSS_IOS_SDK/OSSService.h>
```

Initialization

Before using the SDK, you must get the OSS Service in the SDK and the initialize some settings, such

as the signature method and the domain name of the default data center.

In the whole application lifecycle, you only have to initialize these items once before using the OSS iOS SDK.

1. Getting the OSS Service

The OSS iOS SDK uses service IDs to provide various functions. The retrieval method is as follows:

```
id<ALBBOSSServiceProtocol> ossService = [ALBBOSSServiceProvider getService];
```

In the application’ s lifecycle, you can get this ossService to use the OSS service multiple times.

2. Setting the Data Center Domain Name

When creating a bucket on the OSS official site, you can choose a data center based on the unit price, the distribution of request sources, and the response latency. When you create a bucket, if you do not specify its data center, OSS will automatically allocate it a default data center. The current default data center is oss-cn-hangzhou.

Therefore, when performing OSS data operations, you must use a domain name to specify the data center of your bucket. This can be done through the following interface:

```
[ossService setGlobalDefaultBucketHostId:@"oss-cn-qingdao.aliyuncs.com"]; // Specifies that your bucket is placed in the Qingdao data center
```

If you **have not** called this interface to set the domain name, the OSS iOS SDK will set your hostId to ‘oss-cn-hangzhou.aliyuncs.com’ by default.

3. Token Generator Settings

For detailed information concerning token generator settings, refer to the [Access Control] chapter in this document. For now, you only need to know that, during initialization, you must perform this operation.

```
[ossService setAuthenticationType:ORIGIN_AKSK];
[ossService setGenerateToken:^(NSString *method, NSString *md5, NSString *type, NSString *date, NSString *xoss, NSString *resource){
// Requires specific implementation
}];
```

4. Custom Reference Time Setting

Because OSS token verification is time-sensitive, you may worry that incorrect system times of mobile terminals may cause users to be unable to access the OSS service. We have prepared an interface that allows you to set the SDK time. Over the network, you can get the current epoch time from the business server and set it. Then, during SDK operations, the time will be synced with the server time:

```
NSTimeInterval currentEpochTimeInSec = 1429605946.0;
[ossService setCustomStandardTimeWithEpochSec:currentEpochTimeInSec]; // The epoch time is counted in seconds from January 1, 1970 00:00:00 UTC
```

Note that the time uses units of seconds.

Access control

1. Original AK/SK authentication

After activating the OSS service on the Alibaba Cloud official site and creating your storage space (bucket), you can use the OSS iOS SDK to access data on a terminal. To ensure the security of your data, the OSS has made the server reliably secure. Accordingly, you need to implement a corresponding authentication process before you can successfully access your data.

When you register OSS, the system will assign you an Access Key ID and Access Key Secret pair, which is called the ID pair and referred to in this document as ‘AK/SK’. This is used for signature verification when you access OSS. For details on AK/SK usage, refer to User Signature Authentication. Simply put, each request you initiate to the OSS must carry the token derived from the ‘AK/SK’ signature for the requested content. After successful authentication, the OSS server will process your request.

Because you are performing development on a terminal, the AK/SK must be securely stored. Therefore, the OSS iOS SDK will not ask for your ‘AK/SK’. Rather, the SDK will call your own signature method to generate a token before sending a request to the OSS.

You must call the following method:

```
id<ALBBOSSServiceProtocol> ossService = [ALBBOSSServiceProvider getService];
[ossService setAuthenticationType:ORIGIN_AKSK]; // Sets the authentication type
[ossService setGenerateToken:tokenGenerator];
```

Note: The sample code below does not repeat the capability of getting ossService

tokenGenerator is a block that must be activated by the user. Its parameters correspond exactly to the Authorization field calculation method in the OSS API Manual. You just need to generate and return a Token by signing these parameters with AK/SK.

```
```

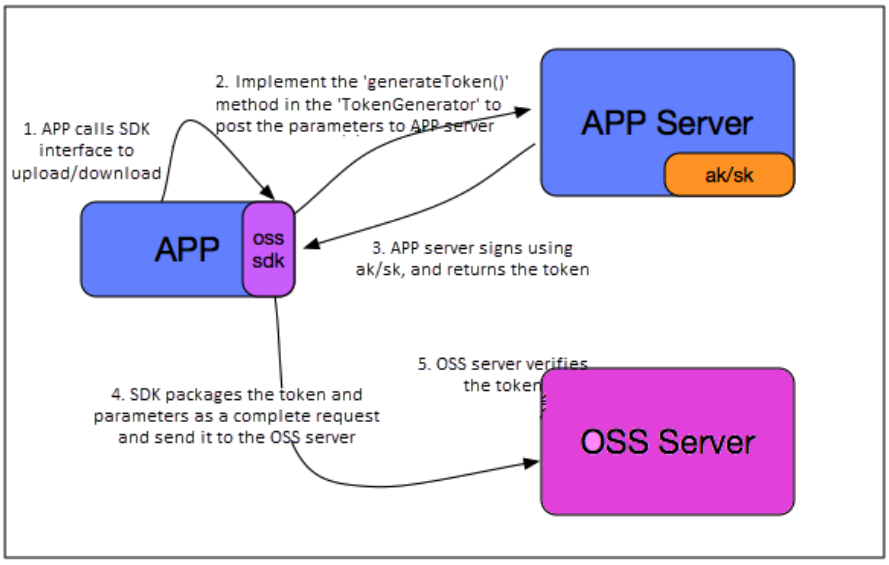
```
@property (nonatomic, strong)NSString * (^generateToken)(NSString *, NSString *, NSString *, NSString *, NSString *, NSString *);
```

If we do not consider the security of the ‘AK/SK’, a sample code is as follows:

```
id<ALBBOSSServiceProtocol> ossService = [ALBBOSSServiceProvider getService];
NSString *accessKey = @"<your accessKey>"; // In actual use, the AK/SK is not stored in the code using plain text
NSString *secretKey = @"<your secretKey>";
[ossService setGenerateToken:^(NSString *method, NSString *md5, NSString *type, NSString *date, NSString *xoss, NSString *resource) {
NSString *signature = nil;
NSString *content = [NSString stringWithFormat:@"%@\n%@\n%@\n%@\n%@%@", method, md5, type, date, xoss, resource];
signature = [OSSTool calBase64Sha1WithData:content withKey:secretKey];
signature = [NSString stringWithFormat:@"OSS %@:%@", accessKey, signature];
NSLog(@"Signature: %@", signature);
return signature;
}];
```

The ‘AK/SK’ should not be stored in the code using clear text. You should design a secure storage method for receiving the ‘AK/SK’.

A suggested programming model is shown below:



As shown in the figure, you can store the ‘AK/SK’ on your business server and then implement the authorized signature method in the ‘TokenGenerator’ to post these parameters to your business server using an HTTP request. When your business server signs and returns the token, you then return this token to the ‘TokenGenerator’.

Therefore, in the entire process, the ‘AK/SK’ will not be exposed on the terminal, effectively avoiding the risk of leaks.

It is difficult to deliver security, ease-of-use, and efficiency at the same time. Before deciding on an AK/SK storage solution, please carefully assess the related risks and benefits.

2. Federation token authentication

As described above, following the recommended security model and adopting the original AK/SK signature requires the terminal to get a signature result from the business server each time it initiates a request to the OSS service. When the terminal needs to access data frequently, the cost of this method will be quite high because a network interaction occurs each time the terminal gets a signature result.

For mobile terminals to conveniently and securely use the OSS service, Alibaba Cloud has developed the STS service to support the security requirements in mobile scenarios. For details, refer to STS API Reference.

The OSS download package already contains STS Client SDKs for different programming languages, allowing the user to conveniently call the STS service on the business server. For the relevant usage, refer to the demo in the package.

In addition, we have provided a simple Android network disk demo based on STS. The source code is hosted on github and provided for reference: sts-demo-simple-android-app

In STS, Federation Token is a core concept. It represents one-time authorization you, as an ISV, can grant to terminal users. In fact, this authorization process requires your original AK/SK (see section one in this chapter).This authorization specifies the permission range and validity period. In the validity period, the Security Token can be used to directly sign requests in the permission range. This validity period generally covers the entire lifecycle of a mobile application, helping completely solve the problem described above. When required, it can also be updated.

The use of the original AK/SK to get the Federation Token and the OSS iOS SDK behavior are independent from each other. The SDK provides an interface for accepting the token, but it is independent of token details. You must call the following interface:

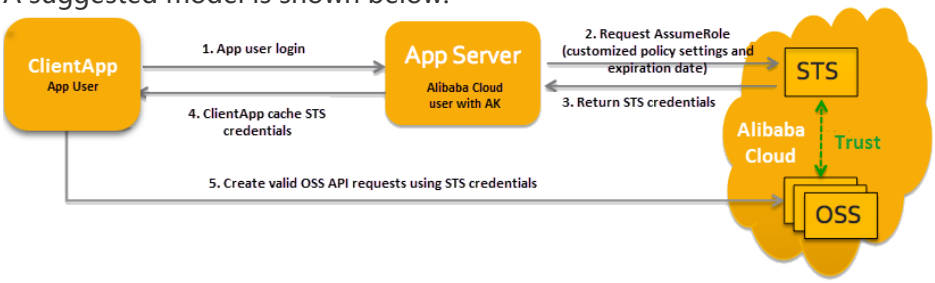
```
[ossService setAuthenticationType:FEDERATION_TOKEN];
[ossService setFederationTokenGetter:(OSSFederationToken *(^)())tokenGetter];
```

First set the ossService authentication method as Federation Token. Then, set the STS token generator in the ossService. The implementation of this token generator depends on your STS Token retrieval method. This is the same as AK/SK authentication. The difference is that the STS Federation Token can be used throughout the validity period. Therefore, in the validly period, the SDK will continue to use this token internally until it expires. Then, it will call this interface again to get another one.

```
[ossService setAuthenticationType:FEDERATION_TOKEN];
[ossService setFederationTokenGetter:^{
// Write your code for getting a new STS Federation Token here
// Under normal conditions, this token should be obtained through a network request to your business server
// Note that the returned OSSFederationToken must include four valid
fields:tempAK/tempSK/securityToken/expiration
// The expiration value is the FederationToken's expiration time, which takes the format of UNIX Epoch time. This is
```

```
the number of seconds from January 1, 1970 00:00:00 in coordinated universal time
}];
```

A suggested model is shown below:



As shown in the figure, the STS service provides servers with SDKs in multiple programming languages. This allows you to conveniently get the Federation Token from your business server and grant it to mobile terminals.

Bucket Operations

1. Bucket settings

Buckets are the namespaces in OSS. They are also the management entities for billing, permission control, logging, and other advanced functions. Before performing data-related operations, you must create and configure a bucket instance. This allows you to easily use this bucket instance to specify data storage locations in subsequent data operations.

Creating a bucket is very simple:

```
OSSBucket *ossBucket = [serviceProvider getBucket:@"oss-example"];
```

Then, configure some optional settings.

1.1 Stating Bucket Access Permissions

When creating a bucket on the OSS official site, you can specify three access permissions: PUBLIC_READ_WRITE, PUBLIC_READ, and PRIVATE. Therefore, in the SDK, you also need to specify the permissions for the bucket you wish to access. This statement will not change your console settings.

```
[sampleBucket setAcl:PRIVATE]; // States the access permission for this bucket
```

1.2 Setting the Data Center Domain Name or CNAME

This can specify the bucket's data center separately or define the domain name already bound to the

bucket:

```
[sampleBucket setOssHostId:@"oss-cn-hangzhou.aliyuncs.com"]; // Specifies the domain name of the bucket's data center or the CNAME domain name already bound to the bucket
```

1.3 Setting CDN Domains

Buckets can separately define the CNAME of the CDN domain they are directed to.Because CDN does not currently support nearest upload, **if you wish to set this, it can only be used during download**. For example:

```
[sampleBucket setCdnAccelerateHostId:@"cname.to.cdn.domain.com"]; // Sets the CNAME domain name for the CDN domain directed to
```

2. Initialization Overview

If you configure all settings when initializing a bucket, the code may be as follows:

```
OSSBucket *ossBucket = [serviceProvider getBucket:@"oss-example"];
[sampleBucket setAcl:PRIVATE]; // Specifies the access permission for this bucket
[sampleBucket setOssHostId:@"oss-cn-hangzhou.aliyuncs.com"]; // Specifies the domain name of the bucket's data center or the CNAME domain name already bound to the bucket
[sampleBucket setCdnAccelerateHostId:@"cname.to.cdn.domain.com"]; // Sets the CNAME domain name for the CDN domain directed to
```

Normally, buckets are global because, in most cases, all data operations may be associated with a bucket. Therefore, you should create these buckets during global initialization. During subsequent data operations, you only need to import the corresponding bucket and specify the data location.

3. List Objects

After setting a bucket, you can call the following interface to list the objects in the bucket:

```
- (ListObjectResult *)listObjectsInBucket:(ListObjectOption *)opt error:(NSError **)error;
```

ListObjectOption is an optional parameter, used to set the Max-Keys, Marker, Prefix, and Delimiter parameters. The significance of these parameters is consistent with the descriptions in **OSS API Manual**. ListObjectResult is the returned result from which you can get the relevant information. For example:

```
ListObjectOption *opt = [[ListObjectOption alloc] init];
[opt setDelimiter:@"/"];
[opt setPrefix:@"prefixdir/"];
```

```
[opt setMaxKeys:500];
[opt setMarker:@"prefix187"];

NSError *error = nil;
ListObjectResult *result = [sampleBucket listObjectsInBucket:opt error:&error]; // Calls the interface to list bucket objects

int objectNum = result.objectList.count;
NSString *nextMarkder = result.nextMarker;
BOOL isTruncated = result.isTruncated;
NSArray *objectList = result.objectList; // Gets the objects list from the returned results

for (ObjectInfo *ele in objectList) { // Traverses the returned objects list
    NSString *objectKey = ele.objectKey;
    NSString *lastModifiedTime = ele.lastModified;
}

NSMutableArray *commonPrefixs = result.commonPrefixList; // When Delimiter is specified, objects with the same prefix will be grouped together and recorded in the returned commonprefix
for (NSString *Prefix in commonPrefixs) {
    // doSomething with prefixed
}
```

Note: This method is a synchronous method. When executed, the method will send network requests. Therefore, do not directly call it in the main thread.

Data Storage

The OSS iOS SDK performs data storage operations using 'OSSData'. Data here indicates the data in the memory when an application is running. Therefore, if you need to upload a data segment in the memory when the application is running or wish to download a piece of data from the OSS to the local memory as 'NSData' for processing, you should perform such an operation.

When constructing an 'OSSData' instance, you must specify 'OSSBucket' and 'ObjectKey'. This indicates the data item in the OSS you want to work on. The instance is obtained as follows:

```
OSSData *ossData = [ossService getOSSDataWithBucket:ossBucket key:@"<objectName>"];
```

1. Data Downloads

After you construct an 'OSSData' instance, you can download the instance if the 'OSSBucket' and 'ObjectKey' you specify correspond to data existing in the OSS. The code is as follows:

```
NSError *error = nil;
NSData *yourData = [ossData get:&error];//storage error message
```

To download a certain range of data, use the code below:

```
NSError *error = nil;
[ossData setRangeFrom:1 to:40];
[ossData setRangeFrom:30 to:RANGE_INFINITE]; // Sets the download range between 30 and the end of the object
[ossData setRangeFrom:RANGE_INFINITE to:20]; // Indicates a download of the last 20 bytes of the object
NSData *yourData = [ossData get:&error]; // storage error message
```

Here, you must note the following:

- This is a synchronous return and may clog the thread until the method is returned.
- During data operations, the OSS iOS SDK must interwork with the OSS server. Therefore, this code is considered time-consuming and obstructive. However, we provide the asynchronous interface 'getWithDataCallback: withProgressCallback:'. Please refer to the section below.

2. Asynchronous Data Download Version

In fact, the OSS iOS SDK provides an asynchronous version for all time-consuming methods, allowing you to import your own implemented callback method. This performs the operation in a new thread and asynchronously calls back your processing logic.

For data downloads, the asynchronous interface is used as follows:

```
OSSData *testData = [ossService getOSSBucketWithBucket:ossBucket key:@"<objectName>"];

[testData getWithDataCallBack:^(NSData *data, NSError *error) {
    if (error) {
        NSLog(@"Error: %@", error);
        return;
    }
    NSLog(@"Success, data length: %lu", (unsigned long)data.length);
} withProgressCallBack:^(float progress) {
    NSLog(@"Current progress: %f", progress);
}];
```

If you need to specify a data range to download, use the code below:

```
[testData setRangeFrom:1 to:40];

[testData getWithDataCallBack:^(NSData *data, NSError *error) {
    if (error) {
        NSLog(@"Error: %@", error);
        return;
    }
    NSLog(@"Success, data length: %lu", (unsigned long)data.length);
} withProgressCallBack:^(float progress) {
    NSLog(@"Current progress: %f", progress);
}];
```

When using the asynchronous interface, please **be sure to import a callback method**. If you do not need a callback method, you can leave it blank.

3. Data Uploads

If, when running an application, you must upload the data in the memory to the OSS, you must first specify the 'OSSBucket' and 'ObjectKey' to construct an 'OSSData' instance. Then, call the 'setData' interface and specify the data to upload and its type. Finally, call the upload method again to upload the data.

The code is as follows:

```
NSError *error = nil;
OSSData *testData = [ossService getOSSDataWithBucket:ossBucket key:key];
[testData enableUploadCheckMd5sum:YES]; // Enables the upload MD5 check
[testData setData:upData withType:@"<dataType>"];
[testData upload:&error];
```

After the upload is complete, the bucket named 'sampleBucket' on the OSS will have a new data item named 'sampleData', which contains the 'data' you have uploaded.

Likewise, data upload also has an asynchronous version:

```
[testData enableUploadCheckMd5sum:YES]; // Enables the upload MD5 check
[testData setData:upData withType:@"<dataType>"];
[testData uploadWithUploadCallback:^(BOOL isSuccess, NSError *error) {
    if (isSuccess) {
        NSLog(@"Upload success!");
    } else {
        NSLog(@"Error: %@", error);
    }
} withProgressCallback:^(float progress) {
    NSLog(@"Current progress: %f", progress);
}];
```

4. Data Deletion

After you construct an 'OSSData' instance representing the data corresponding to the specified 'OSSBucket' and 'ObjectKey', you can call the delete method to delete this data item. The code is as follows:

```
NSError *error = nil;
[testData delete:&error];
```

Likewise, there is an asynchronous version:

```
```

```
[testData deleteWithDeleteCallback:^(BOOL isSuccess, NSError *error) {
    if (isSuccess) {
        NSLog(@"Delete success!");
    } else {
        NSLog(@"Error: %@", error);
    }
}];
```

The delete operation shows no progress status. You do not need a callback method for the implementation progress. The copy method below is similar.

5. Data Copying

You can copy data from an existing object to the object represented by testData. You just have to specify the data's sourceOSSBucket and sourceObjectName.

The code is as follows:

```
NSError *error = nil;
[testData copyFromBucket:@"<sourceOSSBucket>" key:@"<sourceObjectName>" error:&error];
```

Asynchronous version:

```
[testData copyFromWithBucket:@"<sourceOSSBucket>" withKey:@"<sourceObjectName>"
withCopyCallback:^(BOOL isSuccess, NSError *error) {
    if (isSuccess) {
        NSLog(@"Copy success!");
    } else {
        NSLog(@"Error: %@", error);
    }
}];
```

6. Generating Data URLs

After constructing the data for the specified 'OSSBucekt' and 'objectKey', you can call the 'getResourceURL' interface to generate an access URL. This can be used to authorize URL access by third-parties. If the data's bucket does not have 'Public-Read' permission, you must import an 'accessKey' and validity period for this interface. This URL will expire at the end of the validity period.

```
NSString *url = [ossData getResourceURL:@"accessKey" andExpire:availablePeriodInSeconds]; // The generated URL
will expire in availablePeriodInSeconds seconds
```

If your bucket has the 'Public-Read' permission, you just have to call:

```
NSString *url = [ossData getResourceURL]; // Because the Bucket has Public-Read permission, you do not need to
import an accessKey or validity period
```

7. Cancelling Asynchronous Uploads/Downloads

If, for some reason, you do not wish to continue an asynchronous upload/download operation, you just need to call this interface to obtain a 'taskHandler' and then call '[taskHandler cancel]', as shown below:

```
TaskHandler * taskHandler = [testData getWithDataCallBack:^(NSData *data, NSError *error) {
    if (error) {
        NSLog(@"Error: %@", error);
        return;
    }
    NSLog(@"Success, data length: %lu", (unsigned long)data.length);
} withProgressCallBack:^(float progress) {
    NSLog(@"Current progress: %f", progress);
}];

[taskHandler cancel];
```

Note: The operation for cancelling asynchronous file uploads/downloads is similar and will not be repeated in the following sections.

8. Adding Custom Meta Attributes During Uploads

Data in the OSS has its own meta attributes. In addition to attributes automatically added by the system, you can specify custom attributes using the "x-oss-meta-" prefix. For details, please refer to the OSS product documentation.

The attributes you specify at the time of upload can be individually obtained through the get meta attributes method in the SDK. For details, refer to the [Only Retrieve Meta] section herein.

The code for adding custom meta attributes prior to upload is as follows:

```
[ossData setMetaKey:@"x-oss-meta-key1" withMetaValue:@"value1"]; // This is only valid for upload operations and takes effect when called prior to the upload

[ossData setMetaKey:@"x-oss-meta-key2" withMetaValue:@"value2"]; // Custom meta attributes must have the "x-oss-meta-" prefix

[ossData setMetaKey:@"x-oss-meta-key3" withMetaValue:@"value3"]; // Meta attribute key-value pairs of the same name are not supported
```

Note: If you add meta attributes without the "x-oss-meta-" prefix, they will be ignored.

File Operations

In the OSS iOS SDK, the file operation set is put in the 'OSSFile' class. This class works largely the

same way as 'OSSData'.In fact, the **differences concern the fact that** 'OSSData' can be used to retrieve data in the memory, while 'OSSFile' is used to directly upload local files on the terminal and download OSS data to local files.

As it takes time to upload a large file and the upload may fail halfway due to various exceptions, the OSS iOS SDK provides an interface for resumable data transfer for the upload of large files.

1. Downloading to Files

Downloading to files is similar to data downloading, but requires you to specify the destination file path. The download method will not return data.

```
NSError *error = nil;
OSSFile *testFile = [ossService getOSSFileWithBucket:ossBucket key:@"<objectName>"];
[testFile downloadTo:yourPath error:&error];
```

The code used to specify a data range for download is as follows:

```
NSError *error = nil;
OSSFile *testFile = [ossService getOSSFileWithBucket:ossBucket key:@"<objectName>"];
[testFile setRangeFrom:1 to:0.];
[testFile downloadTo:yourPath error:&error];
```

Asynchronous version:

```
[testFile downloadTo:toPath withDownloadCallback:^(BOOL isSuccess, NSError *error) {
    if (isSuccess) {
        NSLog(@"Download file success!");
    } else {
        NSLog(@"Error: %@", error);
    }
} withProgressCallback:^(float progress) {
    NSLog(@"Current progress: %f", progress);
}];
```

Asynchronous version for specified data ranges:

```
[testFile setRangeFrom:1 to:0.];
[testFile downloadTo:toPath withDownloadCallback:^(BOOL isSuccess, NSError *error) {
    if (isSuccess) {
        NSLog(@"Download file success!"];
    } else {
        NSLog(@"Error: %@", error);
    }
} withProgressCallback:^(float progress) {
    NSLog(@"Current progress: %f", progress);
}];
```

2. Uploading From Files

Before uploading, you must specify the path of the file to be uploaded and its content type.

```
NSError *error = nil;
OSSFile *testFile = [ossService getOSSFileWithBucket:ossBucket key:key];
[testData enableUploadCheckMd5sum:YES]; // Enables the upload MD5 check
[testFile setPath:@"your path" withContentType:@"<fileType>"];
[testFile upload:&error];
```

Asynchronous version:

```
[testFile uploadWithUploadCallback:^(BOOL isSuccess, NSError *error) {
    if (isSuccess) {
        NSLog(@"Upload file success!");
    } else {
        NSLog(@"Error: %@", error);
    }
} withProgressCallback:^(float progress) {
    NSLog(@"Current progress: %f", progress);
}];
```

3. Remaining Common Operations

The two sections above have already described the differences between 'OSSFile' and 'OSSData'. From the perspective of OSS, there is obviously no difference, as they both store data segments in the OSS. Therefore, apart from the differences between the upload/download interfaces in data format, other interfaces work in the same way, e.g. copying and deleting.

Thus, we will not describe them here.

4. Resumable Data Transfer

Because it takes much time to upload a large file, the uploading process is highly prone to failure. Therefore, the OSS Android SDK provides an interface for resumable data transfer. The next call to this interface will continue the upload from the breakpoint of the last operation. If the last upload is successful, data will be uploaded again to overwrite the original target.

This interface is only available in an asynchronous version. The sample code is as follows:

Multipart upload:

```
OSSFile *testFile = [ossService getOSSFileWithBucket:ossBucket key:key];
[testFile setPath:@"your path" withContentType:@"<fileType>"];
[testFile resumableUploadWithUploadCallback:^(BOOL isSuccess, NSError *error) {
    if (isSuccess) {
```

```
        NSLog(@"Upload file success!");
    } else {
        NSLog(@"Error: %@", error);
    }
} withProgressCallback:^(float progress) {
    NSLog(@"Current progress: %f", progress);
}];
```

Retrieving Meta

You can use 'OSSMeta' to independently retrieve data's meta attributes. The collection of retrieved meta attributes are returned to you in the 'NSDictionary' format.

The sample code is as follows:

```
NSError *error = nil;
OSSMeta *meta = [ossService getOSSMetaWithBucket: ossBucket key:key];
NSDictionary *dic = [meta getMeta:&error];
```

Asynchronous version:

```
[meta getWithMetaCallback:^(NSDictionary *head, NSError *error) {
    if (error) {
        NSLog(@"Error code: %ld", (long)[error code]);
        return;
    }
    NSLog(@"Get meta: %@", head);
}];
```

Independent Multipart Uploads

If breakpoint uploads do not meet your needs, the SDK provides an independent multipart upload interface, allowing you to control your upload parts.There are five interfaces associated with multipart uploads: get UploadId, upload a single part, complete upload, list uploaded parts, and cancel upload. These interfaces are used through 'OSSMultipart'. During a multipart upload, you can continue using an 'OSSMultipart' object, or call different 'OSSMultipart' objects when calling each interface. For example:

```
NSString * uploadObjectKey = @"multipartUploadDataKey";
NSError * error = nil;

OSSMultipart * multipart = [service getOSSMultipartWithBucket:bucket key:uploadObjectKey];
```

```
NSString * uploadId = [multipart initiateMultiPartUploadWithError:&error];
NSLog(@"UploadId: %@", uploadId);

NSMutableArray * uploadPartResults = [[NSMutableArray alloc] init];
for (int i = 0; i < 2; i++) { // If you need to upload two parts
    NSData * data = <Data of each part>;
    [multipart setPartNumber:i + 1];
    [multipart setPartData:data];
    UploadPartResult * result = [multipart uploadPartWithError:&error];
    [uploadPartResults addObject:result];
}

NSArray * listPartResult = [multipart listPartsWithError:&error]; // If you need to list the parts that have been
uploaded
for (UploadPartInfo * info in listPartResult) {
    NSLog(@"UploadPartInfo : %d, %@", [info partNumber], [info etag]);
}

[multipart selfSetPartList:uploadPartResults];
[multipart completeMultipartUploadWithError:&error];
```

As you can see, when setting up the data to be uploaded for a single part, it can be accessed from the data stream or you can directly set a byte array as the data. In addition, if you need to use a different 'OSSMultipart' object at each stage to meet your special needs, you must save the information from the previous stage as the upload ID of the next stage. For example:

```
NSString * uploadObjectKey = @"multipartUploadData";
NSError * error = nil;
OSSMultipart * multipart1 = [service getOSSMultipartWithBucket:bucket key:uploadObjectKey];
NSString * uploadId = [multipart1 initiateMultiPartUploadWithError:&error];

NSLog(@"UploadId: %@", uploadId);

NSMutableArray * uploadPartResults = [[NSMutableArray alloc] init];
for (int i = 0; i < 2; i++) {
    OSSMultipart * multipart2 = [service getOSSMultipartWithBucket:bucket key:uploadObjectKey];
    [multipart2 selfSetUploadId:uploadId];
    NSData * data = <Data of each part>;
    [multipart2 setPartNumber:i + 1];
    [multipart2 setPartData:data];
    UploadPartResult * result = [multipart2 uploadPartWithError:&error];
    [uploadPartResults addObject:result];
}

OSSMultipart * multipart3 = [service getOSSMultipartWithBucket:bucket key:uploadObjectKey];
[multipart3 selfSetUploadId:uploadId];
NSArray * listPartResult = [multipart3 listPartsWithError:&error];
for (UploadPartInfo * info in listPartResult) {
    NSLog(@"UploadPartInfo : %d, %@", [info partNumber], [info etag]);
}

OSSMultipart * multipart4 = [service getOSSMultipartWithBucket:bucket key:uploadObjectKey];
[multipart4 selfSetUploadId:uploadId];
[multipart4 selfSetPartList:uploadPartResults];
```

```
[multipart4 completeMultipartUploadWithError:&error];
```

It must be emphasized that all of these interfaces call time-consuming networking operations, and may clog the current thread or cause an error. For error handling, refer to the exception handling chapter below

Exception Handling

The OSS iOS SDK has many interfaces for communication with the OSS server. Each request must be authenticated independently or involves operations on local files. These actions may experience problems. Therefore, the SDK will encapsulate such information into NSError.

In the interface, NSError type parameters are used to store exception information.

Exceptions are classified as follows: local exceptions and OSS exceptions. The former indicates system exceptions, such as network connection failure, IO, file, parameter, or status exceptions. The later indicates that the OSS system could not process a request for reasons like the following: the requested resource does not exist; authentication failed, or an object is too large.

If the NSError domain is marked with an ‘ossException’, an OSS exception occurs. Otherwise, a local exception occurs.

1. Local Exceptions

If the NSError parameter is not null, you can obtain and handle the exception as follows:

```
NSString *errorDomain = [error domain]; // Gets the error domain. For local exceptions, the errorDomain is
@"localException"
NSInteger errorCode = [error code]; // Gets the error code
NSDictionary *errorInfo = error; // Gets the error message
```

2. OSS Exceptions

If the NSError parameter is not null, you can obtain and handle the exception in a similar way you do with local exceptions:

```
NSString *errorDomain = [error domain]; // Gets the error domain. For OSS exceptions, the errorDomain is
@"ossException"
NSInteger errorCode = [error code]; // Gets the error code returned by the OSS server
NSDictionary *errorInfo = error; // Gets the error message returned by the OSS server
```

This exception information is constructed and processed based on the standard error responses in

the OSS API Documentation file.

.NET-SDK

Preface

Introduction

This document introduces the installation and use of the OSS .NET SDK (.NET SDK Version 2015-05-28 particularly).This document assumes that you have already subscribed to the AliCloud OSS service and created an Access Key ID and Access Key Secret.In the document, ID represents the Access Key ID and KEY indicates the Access Key Secret. If you have not yet subscribed to or do not know about the OSS service, please log into the OSS Product Homepage for more help.

SDK download

- SDK: aliyun_dotnet_sdk_2.3.0
- GitHub: click here

Version Revisions

.NET SDK (2015-05-28)

Updates:

2015/05/28

- Added Bucket Lifecycle support. Adding and removing Lifecycle rules allowed;
- Added the DoesBucketExist and DoesObjectExist interfaces for determining the existence of Bucket and Objects;
- Added SwitchCredentials so as to be able to change user account information at runtime;
- Added the ICredentialsProvider interface class to provide a policy for generating customized Credentials through its implementation.
- Added GeneratePostPolicy interface to generate Post Policy ;
- Added asynchronization interface (supporting Put/Get/List/Copy/PartCopy asynchronous operations);
- Added STS support.

- Added custom time calibration function. It can be set through the Client configuration item - SetCustomEpochTicks interface;
- Added support for Chunked encoding transfer. The Content-Length item can be skipped when uploading;
- Fixed the bug of getting null results after setting the Expose Header attribute in Bucket CORS;
- Fixed the bug of SDK only getting the first Prefix out of multiple prefixes contained in CommonPrefixs returned in an ListObjects request.
- Fixed the bug of RequestId and HostId resolved to null in response to an OSS-related exception;
- Fixed the bug of encoding error due to the inclusion of Chinese characters in the source key of the CopyObject/CopyPart interface;

.NET SDK (2015-01-15)

Environment requirements:

- .NET Framework 4.0 and above
- A registered user account on AliCloud.com

Assembly:Aliyun.OSS.dll

Version:1.0.5492.31618

Package structure:

- bin
  - Aliyun.OSS.dll .NET assembly file
  - Aliyun.OSS.pdb debugging and project status information file
  - Aliyun.OSS.xml Assembly comments file
- doc
  - Aliyun.OSS.chm Help file
- src
  - SDK source code
- sample
  - Sample code

Updates:

- Removed OTS branch, assembly renamed to Aliyun.OSS.dll
- .NET Framework version upgraded to 4.0 and above
- OSS: Added interfaces for Copy Part, Delete Objects, Bucket Referer List, etc.
- OSS: Added the ListBuckets pagination function
- OSS: Added CNAME support
- OSS: Fixed Put/GetObject flow interruption problem
- OSS: Added samples

.NET SDK (2014-06-26)

Open Services SDK for .NET included OSS and OTS SDKs. .NETSDK used the same interface design as Java SDK and made some improvements based on C#. (The latest version supports multipart uploads to the OSS)

Environment requirements:

- .NET Framework 3.5 SP1 or above
- A registered user account on AliCloud.com, and a subscription to related services (OSS, OTS).

Updates:2014/06/26- OSS:

- Added CORS functionality.

2013/09/02- OSS:

- Fixed the bug of being unable to throw correct exceptions in some cases.
- Optimized SDK performance.

2013/06/04- OSS:

- Changed the default OSS service access method to a third-level domain name access method.

2013/05/20- OTS:

- Updated the default OTS service address to:http://ots.aliyuncs.com
- Added Mono support.
- Fixed some bugs in the SDK, so that it runs more stably.

2013/04/10- OSS:

- Added Object Multipart Upload functionality.
- Added Copy Object functionality.
- Added the ability to generate pre-signed URLs.
- Isolated the IOSS interface to be inherited by OssClient.

2012/10/10- OSS:

- Updated the default OSS service address to:http://oss.aliyuncs.com

2012/09/05- OSS:

- Resolved the problem of invalid parameters like Prefix for ListObjects.

2012/06/15- OSS:

- Added OSS support for the first time. Included basic operations, such as create, modify, read and delete, on OSS Bucket, ACL and Object.

- OTS:
- OTSClient.GetRowsByOffset supports reverse read.
- Added an automatic error handling mechanism for special requests.
- Added HTML help files.

Installation

Using the SDK Directly from within Visual Studio

Steps:

- Download the OSS .NET SDK from the official website.
- Decompress the file.
- After decompression, copy all the files from the bin folder to your project folder.
- Add reference in the Visual Studio project, and select the Aliyun.OSS.dll file that was copied in the previous step.
- After completing the steps above, you can use the OSS .NET SDK in the project.

Quick Start

In this chapter, you will learn how to use the basic functions of the OSS .NET SDK.

Step-1. Initialize an OSSClient

SDK OSS operations are performed through the OSSClient class. The code below creates an OSSClient object:

```
using Aliyun.OpenServices.OpenStorageService;

const string accessId = "<your access id>";
const string accessKey = "<your access key>";
//Using the Hangzhou node as an example
const string endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// Initialize OSSClient instance
var ossClient = new OssClient(endpoint, accessId, accessKey);
```

In the above code, the variables accessKeyId and accessKeySecret are allocated to the user by the system. They are called the ID pair and used to identify the user. They may be used to perform signature verification when your AliCloud account or RAM account accesses OSS. For more

information on the OSSClient, refer to OSSClient.

Step-2. Create a bucket

Buckets are the OSS global namespace. They are equivalent to a data container and can store objects. You can create a bucket with the following code:

```
public void CreateBucket(string bucketName)
{
    var ossClient = new OssClient(endpoint, accessId, accessKey);
    try
    {
        ossClient.CreateBucket(bucketName);
        Console.WriteLine("Create bucket succeed.");
    }
    catch (OssException ex)
    {
        Console.WriteLine(  "Create bucket failed,   " + ex.Message);
    }
}
```

For bucket naming rules, refer to the naming rules in Bucket.

Step-3. Upload objects

Objects are the basic data units in OSS. You can simply think of them as files. The code below can be used to upload an object:

```
public void PutObject(String bucketName, String key, String fileToUpload)
{
    using (var fs = File.Open(fileToUpload, FileMode.Open))
    {
            var metadata = new ObjectMetadata();
            metadata.UserMetadata.Add("key0", "val0");
            metadata.ContentLength = fs.Length;
            var result = client.PutObject(bucketName, key, fs, metadata);
            Console.WriteLine(result.Etag);
    }
}
```

The object is uploaded to OSS in InputStream form. In the above example, we can see that, each time you upload an object, you must specify the ObjectMetadata associated with it. ObjectMetaData is the user's description of the object, composed of a series of name-value pairs. Here, ContentLength is required for the SDK to correctly identify the size of the object to be uploaded. In order to ensure the consistency between files uploaded to the server and local files, users can set up ContentMD5. OSS will calculate and compare the MD5 value for the upload data with the MD5 value uploaded by the user. If they are inconsistent, the system will return the InvalidDigest error code. For object naming

rules, refer to the naming rules in Object. For more information on uploading objects, refer to Uploading Objects in Object.

Step-4. List all objects

When you complete a series of uploads, you may need to view which objects are in a bucket. This can be done with the following program:

```
public void ListObject(String bucketName)
{
    var listObjectsRequest = new ListObjectsRequest(bucketName);
    var result = ossClient.ListObjects(listObjectsRequest);
    foreach (var summary in result.ObjectSummaries)
    {
    Console.WriteLine(summary.Key);
    }
}
```

Step-5. Retrieve a specified object

You can refer to the code below to easily retrieve an object:

```
public void GetObject(String bucketName, string key)
{
    var object = ossClient.GetObject(bucketName, key);
    using (var requestStream = object.Content)
    {
      byte[] buf = new byte[1024];
      var fs = File.Open(fileToDownload, FileMode.OpenOrCreate);
      var len = 0;
      while ((len = requestStream.Read(buf, 0, 1024)) != 0)
      {
          fs.Write(buf, 0, len);
      }
      fs.Close();
}
```

When you call the OSSClient's getObject method, it returns an OSSObject object, containing various object information. Using the OSSObject's getObjectContent method, you can retrieve the returned object input stream and obtain the object content by reading it. When you are finished, close the stream.

OssClient

The OSSClient is the C# client of OSS services. It provides a series of functions to be called for

interaction with OSS services.

Creating the OSSClient

It is very easy to create an OSSClient, as shown in the code below:

```
const string accessId = "<your access id>";
const string accessKey = "<your access key>";
// Using Hangzhou OSS Endpoint as an example
const string endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// Initialize OSSClient instance
var ossClient = new OssClient(endpoint, accessId, accessKey);
```

It is critical that the user import the AccessKey and access the bucket endpoint.

Configuring the OSSClient

To configure detailed parameters for the OSSClient, you can import a ClientConfiguration object when creating the OSSClient. ClientConfiguration is the OSS service configuration type. It allows you to configure a proxy for the client, the maximum number of connections, and other parameters.

Setting Network Parameters

We can use ClientConfiguration to set some network parameters:

```
conf.ConnectionLimit = 512;  // Maximum number of concurrent HttpWebRequest connections
conf. MaxErrorRetry = 3; //Maximum number of retries upon a Set request error
conf. ConnectionTimeout = 300; //Connection timeout time
conf. SetCustomEpochTicks(customEpochTicks); //Set custom reference time
```

Using a Proxy

The code below allows the client to use a proxy to access the OSS service:

```
// Creates a ClientConfiguration instance
ClientConfiguration conf = new ClientConfiguration();

// Configures the proxy for the local port 8080
conf.ProxyHost = "127.0.0.1";
conf.ProxyPort = 8080;

// Creates the OSSClient
client = new OssClient(endpoint, accessKeySecret, accessKeySecret, conf);
```

Bucket

OSS uses buckets as namespaces for user files and also as the management entities for advanced functions such as billing, permission control, and logging. The bucket name must be globally unique in the entire OSS and cannot be changed. Every object stored on OSS must be included in a bucket. One application, such as an image sharing website, corresponds to one or multiple buckets. A user can create a maximum of 10 buckets, but there is no limit on the number of objects in each bucket. Each bucket can store up to 2 PB of data.

Naming Rules

The bucket naming rules are as follows:

- It can only contain lower-case letters, numbers, and dashes (-).
- It must start with a lower-case letter or number.
- The length must be 3-63 bytes

Creating Buckets

We can use the code below to create a bucket:

```
String bucketName = "my-bucket-name";
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);
// Creates a Bucket
client.CreateBucket(bucketName);
```

Because bucket names are globally unique, do your best to ensure your bucket names are not the same as other people's.

Listing all Buckets of a User

The code below will list all the buckets of the user:

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

var buckets = client.ListBuckets();
foreach (var bucket in buckets)
{
    Console.WriteLine(bucket.Name + ", " + bucket.Location + ", " + bucket.Owner);
}
```

CNAME Access

After a user directs his own domain name's CNAME to his bucket's domain name, he can access OSS through his domain name:

```
// For example, your domain name is "http://my-cname.com" and you direct the CNAME to your bucket domain name "mybucket.oss-cn-hangzhou.aliyuncs.com"
OssClient client = new OssClient("http://my-cname.com/", /* accessKeyId */, /* accessKeySecret */);
PutObjectResult result = client.putObject("mybucket", /* key */, /* input */, /* metadata */);
```

Users just need to create an OSSClient class instance and change the endpoint originally entered in the bucket to the post-CNAME domain name. At the same time, users must note that, when using this OSSClient instance for subsequent operations, the bucket name can only be filled by the indicated bucket name.

Determining If a Bucket Exists

To determine if a bucket exists, we can use the following code:

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

String bucketName =  “your-bucket” ;
var exist = client.DoesBucketExist(bucketName);

Console.WriteLine("exist ? " + exist);
```

Setting Bucket ACL

To set the bucket ACL, we can use the following code:

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

String bucketName =  “your-bucket” ;
// Make bucket ACL as public read
client.SetBucketAcl(bucketName, CannedAccessControlList.PublicRead);
```

Retrieving Bucket ACL

To retrieve the bucket ACL, we can use the following code:

```
// Initialize OSSClient
```

```
var client = new OssClient(endpoint, accessId, accessKey);

String bucketName =  “your-bucket” ;
var acl = client.GetBucketAcl(bucketName);

Console.WriteLine(acl.ToString());
```

Deleting Buckets

The following code deletes a bucket:

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

String bucketName =  “your-bucket” ;

client.DeleteBucket(bucketName);
```

You must note that, if the bucket is not empty (the bucket contains objects), it cannot be deleted. You must delete all objects in a bucket before deleting the bucket.

Object

In OSS, objects are the basic data units for user operation.The maximum size of a single object may vary depending on the data uploading mode. The size of an object cannot exceed 5 GB in the Put Object mode or 48.8 TB in the multipart upload mode. An object includes the key, meta, and data. The key is the object name; meta is the user's description of the object, composed of a series of name-value pairs; and data is the object data.

Naming Rules

Object naming rules:

- It uses UTF-8 encoding
- The length must be 1-1023 bytes
- It cannot start with "/" or "\"
- It cannot contain "\r" or "\n" line breaks

Uploading Objects

Simple Upload

Upload the specified string:

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

string str = "a line of simple text";
byte[] binaryData = Encoding.ASCII.GetBytes(str);
MemoryStream requestContent = new MemoryStream(binaryData);
client.PutObject(bucketName, key, requestContent);
```

Upload the specified local file

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

String fileToUpload =  “your local file” ;
client.PutObject(bucketName, key, fileToUpload);
```

The largest file to upload using this method cannot exceed 5 GB. If the size limit is exceeded, you may use multipartupload to upload instead.

Creating Simulated Folders

The OSS service does not use folders. All elements are stored as objects. However, users can create simulated folders using the following code:

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

// Note: key treats as a folder and must end with a slash.
const string key = "yourfolder/";
// put object with zero bytes stream.
using (MemoryStream ms = new MemoryStream())
{
client.PutObject(bucketName, key, ms);
}
```

Creating a simulated folder is in fact creating an object with a size of 0. This object can still be uploaded and downloaded. The console will display any object ending with "/" as a folder. Therefore, users can create simulated folders this way. For accessing folders, refer to the folder simulation function

Setting the Object's Http Header

The OSS service allows users to customize the object Http Header. The following code sets the expiration time for the object:

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);
```

```
using (var fs = File.Open(fileToUpload, FileMode.Open))
{
    var metadata = new ObjectMetadata();
    metadata.ContentLength = fs.Length;
    metadata.ExpirationTime = DateTime.Parse("2015-10-12T00:00:00.000Z")
    client.PutObject(bucketName, key, fs, metadata);
}
```

The .NET SDK supports four types of Http Headers:Cache-Control, Content-Disposition, Content-Encoding, and Expires.For details on the headers, please see RFC2616.

User-Defined Metadata

The OSS allows users to define metadata to describe the object. For example:

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

using (var fs = File.Open(fileToUpload, FileMode.Open))
{
    var metadata = new ObjectMetadata();
    metadata.UserMetadata.Add("name", "my-data");
    metadata.ContentLength = fs.Length;
    client.PutObject(bucketName, key, fs, metadata);
}
```

In the above code, the user has defined a metadata with its name as "name" and its value as "my-data". When downloading this object, users will also obtain the metadata.A single object can have multiple similar parameters, but the total size of all user meta cannot exceed 2 KB.

> NOTE:The user meta name is not case sensitive. For instance, when a user uploads an object and defines the metadata name as "Name", the parameter stored in the header will be: "x-oss-meta-name". Therefore, when accessing the object, just use parameters named "name". However, if the stored parameter is "name", and no information can be found for the parameter, the system will return "Null"

Multipart Upload

OSS allows users to split an object into several requests for uploading to the server. Concerning the multipart upload content, refer to the Object Multipart Upload section in MultipartUpload.

List Bucket Objects

Listing Objects

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

public void ListObject(String bucketName)
{
    var listObjectsRequest = new ListObjectsRequest(bucketName);
    var result = ossClient.ListObjects(listObjectsRequest);
    foreach (var summary in result.ObjectSummaries)
    {
        Console.WriteLine(summary.Key);
    }
}
```

NOTE:By default, if a bucket contains more than 100 objects, the first 100 will be returned and the IsTruncated parameter in the returned results will be true. The returned NextMarker can be used as the start point for next data access.The number of object entries returned can be increased by modifying the MaxKeys parameter or using the Marker parameter for separate access.

Extended Parameters

Generally, the ListObjectsRequest parameter provides more powerful functions. For example:

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

var listObjectsRequest = new ListObjectsRequest(bucketName)
{
  Delimiter =  "/" ,
  Marker =  "abc"
};
result = ossClient.ListObjects(listObjectsRequest);

foreach (var summary in result.ObjectSummaries)
{
  Console.WriteLine(summary.Key);
}
```

Settable parameter names and their function:

Name	Function
Delimiter	Used to group object name characters. All names contain the specified prefix and the object between the first Delimiter characters acts as a group element: CommonPrefixes.
Marker	Sets up the returned results to begin from the first entry after the Marker in alphabetical order.
MaxKeys	Limits the maximum number of objects returned for one request. If not specified, the

	default value is 100. The MaxKeys value cannot exceed 1000.
Prefix	requires the returned object key to be prefixed with prefix. Note that the keys returned from queries using a prefix will still contain the prefix.

Multiple iterations must be performed to traverse a whole batch of over 1,000 objects. During each iteration, the final object key of the last iteration can be used as the Marker in the current iteration.

Folder Function Simulation

We can use a combination of Delimiter and Prefix to simulate folder functions.Combinations of Delimiter and Prefix serve the following purposes:Setting Prefix as the name of a folder enumerates the files starting with this prefix, recursively returning all files and subfolders in this folder.When the Delimiter is set as "/", the returned values will enumerate the files in the folder and the subfolders will be returned in the CommonPrefixes section. Recursive files and folders in subfolders will not be displayed. If the bucket contains 4 files: oss.jpg, fun/test.jpg, fun/movie/001.avi, and fun/movie/007.avi. We use the "/" symbol as the separator for folders.

List All Bucket Files

To retrieve all files in a bucket, write the following:

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

ObjectListing result = null;
string nextMarker = string.Empty;
do
{
    var listObjectsRequest = new ListObjectsRequest(bucketName)
    {
        Marker = nextMarker,
        MaxKeys = 100
    };
    result = client.ListObjects(listObjectsRequest);
    foreach (var summary in result.ObjectSummaries)
    {
        Console.WriteLine(summary.Key);
    }
    nextMarker = result.NextMarker;
} while (result.IsTruncated);
```

Output:

```
Objects:
fun/movie/001.avi
```

```
fun/movie/007.avi
fun/test.jpg
oss.jpg

CommonPrefixs:
```

Recursively List All Files in a Directory

We can set the Prefix parameter to retrieve all the files under a directory:

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

var listObjectsRequest = new ListObjectsRequest(bucketName)
{
    Prefix = “fun/”
};
result = client.ListObjects(listObjectsRequest);
foreach (var summary in result.ObjectSummaries)
{
   Console.WriteLine(summary.Key);
}
```

Output:

```
Objects:
fun/movie/001.avi
fun/movie/007.avi
fun/test.jpg

CommonPrefixs:
```

List Files and Subdirectories in a Directory

Using Prefix and Delimiter together, we can list the files and subdirectories under a directory:

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

var listObjectsRequest = new ListObjectsRequest(bucketName)
{
    Prefix = “fun/” ,
    Delimiter = “/”
};
result = client.ListObjects(listObjectsRequest);
Console.WriteLine("Objects:");
foreach (var summary in result.ObjectSummaries)
{
   Console.WriteLine(summary.Key);
}
```

```
Console.WriteLine("CommonPrefixes:");
foreach (var prefix in result.CommonPrefixes)
{
    Console.WriteLine(prefix);
}
```

Output:

```
Objects:
fun/test.jpg

CommonPrefixs:
fun/movie/
```

In the returned results, the ObjectSummaries list contains the files in the fun directory.The CommonPrefixs list shows all subfolders in the fun directory.Obviously, the files fun/movie/001.avi and fun/movie/007.avi are not listed, because they are in the movie directory under the fun folder.

Retrieving Objects

Simply Getting Object

The following code can be used to get and input an object into a stream:

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

public void GetObject(String bucketName, string key)
{
var o = client.GetObject(bucketName, key);
using (var requestStream = o.Content)
{
  byte[] buf = new byte[1024];
  var fs = File.Open(fileToDownload, FileMode.OpenOrCreate);
  var len = 0;
  while ((len = requestStream.Read(buf, 0, 1024)) != 0)
  {
      fs.Write(buf, 0, len);
  }
  fs.Close();
}
}
```

OSSObject contains various object information, including the object's bucket, object name, metadata, and an input stream. The input stream can be used to get and store the object content into an object or the memory.ObjectMetadata contains the ETag, Http Header, and custom metadata defined when the object was uploaded.

Using GetObjectRequest to Retrieve Objects

For more functions, we can use GetObjectRequest to retrieve objects.

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

public void GetObject(String bucketName, string key)
{
    var getObjectRequest = new GetObjectRequest(bucketName, key);
    getObjectRequest.SetRange(20, 100);
    var o = client.GetObject(getObjectRequest);
    using (var requestStream = o.Content)
{
   byte[] buf = new byte[1024];
   var fs = File.Open(fileToDownload, FileMode.OpenOrCreate);
   var len = 0;
   while ((len = requestStream.Read(buf, 0, 1024)) != 0)
   {
       fs.Write(buf, 0, len);
   }
   fs.Close();
}
}
```

We can use the setRange method in getObjectRequest to return the object range. We can use this function for multipart downloads of files and resumable data transfer. GetObjectRequest can set the following parameters:

Parameter	Description
Range	Specifies the range of file transfer.
ModifiedSinceConstraint	If the specified time is earlier than the actual modification time, the file is transmitted normally.Otherwise, the system throws the 304 Not Modified exception.
UnmodifiedSinceConstraint	If the specified time is the same as or later than the actual modification time, the file is transmitted normally.Otherwise, the system throws the 412 precondition failed exception
MatchingETagConstraints	Enters an ETag group. If the entered expected ETag matches the object's ETag, the file is transmitted normally.Otherwise, the system throws the 412 precondition failed exception
NonmatchingEtagConstraints	Enters an ETag group. If the entered expected ETag does not match the object's ETag, the file is transmitted normally.Otherwise, the system throws the 304 Not Modified exception.
ResponseHeaderOverrides	Customizes some headers in the OSS return request.

Only Retrieve ObjectMetadata

The getObjectMetadata method can be used only to get the ObjectMetadata, instead of the object entity. The code is as follows:

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

public void GetObjectMetadata(String bucketName, string key)
{
    var metadata = client.GetObjectMetadata(bucketName, key);
    ...
}
```

Deleting Objects

The following code deletes an object:

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

public void DeleteObject(String bucketName, string key)
{
    client.DeleteObject(bucketName, key);
    ...
}
```

The following code batch deletes objects:

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

var keys = new List<string>();
var listResult = client.ListObjects(bucketName);
foreach (var summary in listResult.ObjectSummaries)
{
  keys.Add(summary.Key);
}
var request = new DeleteObjectsRequest(bucketName, keys, false);
client.DeleteObjects(request);
```

Copying Objects

Within the same region, you can copy an object on which you have operation permissions.We would like to remind users that, when copying an object larger than 1G, we suggest using the Upload Part Copy method.

Copying One Object

Using the copyObject method, we can copy a single object. The code is as follows:

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

var metadata = new ObjectMetadata();
metadata.AddHeader("mk1", "mv1");
var req = new CopyObjectRequest(sourceBucket, sourceKey, targetBucket, targetKey)
{
  NewObjectMetadata = metadata
};
var ret = client.CopyObject(req);
```

Objects copied by using this method must be smaller than 1 GB, otherwise the system will return an error. If the object is larger than 1 GB, use the Upload Part Copy method given below.

Modifying Object Meta

Copying data can modify the meta information of an existing object. If the address of the copied source object is the same as the address of the destination object, the source object's meta information will be replaced.

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

var metadata = new ObjectMetadata();
metadata.ContentType =  “text/html” ;
var req = new CopyObjectRequest(sourceBucket, sourceKey, sourceBucket, sourceKey)
{
  NewObjectMetadata = metadata
};
var ret = client.CopyObject(req);
```

Multipart Upload

Besides using the putObject interface to upload files to OSS, the OSS also provides a Multipart Upload mode.You can apply the Multipart Upload mode in the following scenarios (but not limited to the following):

- Where breakpoint uploads are needed.
- Uploading an object larger than 100MB.
- In poor network conditions, when the connection with the OSS server is frequently broken.
- When, before uploading the file, you cannot determine its size.

Below, we will give a step-by-step introduction to Multipart Upload.

Step-By-Step Multipart Upload

Initializing Multipart Upload

We use the initiateMultipartUpload method to initialize a multipart upload task:

```
String bucketName = "your-bucket-name";
String key = "your-key";

// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);
// Starts Multipart Upload
var request = new InitiateMultipartUploadRequest(bucketName, objectName);
var result = client.InitiateMultipartUpload(request);
// Prints UploadId
Console.WriteLine(result.UploadId);
```

We use InitiateMultipartUploadRequest to specify the name and bucket of the object to upload.In InitiateMultipartUploadRequest, you can also set the ObjectMetadata, but are not required to specify the ContentLength. The initiateMultipartUpload returned result includes the UploadId. This is the unique identifier of a multipart upload task. We will use this in subsequent operations.

Next, we can use two methods for uploading parts: use Upload Part to upload from the local disk or use Upload Part copy to get a copy of an object from a bucket.

Upload Part Local Upload

Next, we will upload the local file part by part. Let us assume that there is one file in the path /path/to/file.zip. Because it is large, we want to multipart upload it to OSS.

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

var fi = new FileInfo(fileToUpload);
var fileSize = fi.Length;
var partCount = fileSize / partSize;
if (fileSize % partSize != 0)
{
    partCount++;
}
var partETags = new List<PartETag>();
for (var i = 0; i < partCount; i++)
{
    using (var fs = File.Open(fileToUpload, FileMode.Open))
    {
```

```
        var skipBytes = (long)partSize * i;
        fs.Seek(skipBytes, 0);
        var size = (partSize < fileSize - skipBytes) ? partSize : (fileSize - skipBytes);
        var request = new UploadPartRequest(bucketName, objectName, uploadId)
        {
            InputStream = fs,
            PartSize = size,
            PartNumber = i + 1
        };
        var result = _ossClient.UploadPart(request);
        partETags.Add(result.PartETag);
    }
}
```

The main idea of this program is to call the uploadPart method to upload each part. However, you must note the following:

- In the uploadPart method, all parts except the last one must be larger than 100KB.However, the Upload Part interface does not immediately verify the size of the uploaded part (because it does not know whether the part is the last one). It verifies the size of the uploaded part only when Multipart Upload is completed.
- OSS will put the MD5 value of the part data received by the server in the ETag header and return it to the user.
- In order to ensure that the data transmitted over the network is free from errors, users can set ContentMD5. OSS will calculate the MD5 value for the uploaded data and compare it with the MD5 value uploaded by the user. If they are inconsistent, the system will return the InvalidDigest error code.
- The part number range is 1~10000. If you exceed this range, OSS will return the InvalidArgument error code.
- When each part is uploaded, it will take the stream to the location corresponding to the start of the next part.
- After each part is uploaded, the OSS returned results will include the PartETag object. This is the combination of the ETag and PartNumber of the uploaded part. This will be used in subsequent steps, so we need to save it.Generally, we will save these PartETag objects in the List.

Upload Part Copy

Using Upload Part Copy, we copy data from an existing object to upload an object.When copying an object larger than 500MB, we suggest using the Upload Part Copy method.

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

var metadata = client.GetObjectMetadata(sourceBucket, sourceObject);
var fileSize = metadata.ContentLength;

var partCount = (int)fileSize / partSize;
```

```
if (fileSize % partSize != 0)
{
    partCount++;
}

var partETags = new List<PartETag>();
for (var i = 0; i < partCount; i++)
{
    var skipBytes = (long)partSize * i;
    var size = (partSize < fileSize - skipBytes) ? partSize : (fileSize - skipBytes);
    var request = new UploadPartCopyRequest(targetBucket, targetObject, sourceBucket, sourceObject, uploadId)
    {
        PartSize = size,
        PartNumber = i + 1,
        BeginIndex = skipBytes
    };
    var result = client.UploadPartCopy(request);
    partETags.Add(result.PartETag);
}
```

The above program calls the uploadPartCopy method to copy each part. The requirements are basically the same as for UploadPart. You must use BeginIndex to locate the position corresponding to the start of the next part to upload. You must also specify the object to copy with SourceKey

Completing Multipart Uploads

Use the code below to complete a multipart upload:

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

var completeMultipartUploadRequest = new CompleteMultipartUploadRequest(bucketName, objectName,
uploadId);
foreach (var partETag in partETags)
{
    completeMultipartUploadRequest.PartETags.Add(partETag);
}
var result = client.CompleteMultipartUpload(completeMultipartUploadRequest);
```

In the code above, the partETags are saved in the partETag list during multipart upload. After OSS receives the part list submitted by the users, it will verify the validity of each data part individually. After all the data parts have been verified, OSS will combine these parts into a complete object.

Canceling Multipart Upload Tasks

We can use the abortMultipartUpload method to cancel multipart upload tasks.

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);
```

```
CompleteMultipartUploadRequest request = new CompleteMultipartUploadRequest(bucketName, objectName,
uploadId);
client.CompleteMultipartUpload(request);
```

Getting All Multipart Upload Tasks in the Bucket

We can use the listMultipartUploads method to retrieve all upload tasks in the bucket.

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

// Gets all upload tasks in the bucket
var request = new ListMultipartUploadsRequest(bucketName);
var multipartUploadListing = client.ListMultipartUploads(request);
// Get event information
var multipartUploads = multipartUploadListing.MultipartUploads;
foreach (var mu : multipartUploads)
{
  Console.WriteLine( “Key:” + mu.Key + “, UploadId: ” + mu.UploadId);
}
var commonPrefixes = multipartUploadListing. CommonPrefixes;
foreach (var prefix : commonPrefixes)
{
  Console.WriteLine( “Prefix:” + prefix);
}
```

NOTE:Under normal conditions, if a bucket contains more than 1000 multipart upload tasks, the first 1000 will be returned and the IsTruncated parameter in the returned results will be false. The returned NextKeyMarker and NextUploadMarker can be used as the next start point to continue reading the data.If the user wishes to increase the number of multipart upload tasks returned, he can modify the MaxUploads parameter or use the KeyMarker and UploadIdMarker parameters for segmented reading.

Getting Information for All Uploaded Parts

We can use the listParts method to retrieve all the uploaded parts of an upload task.

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

var listPartsRequest = new ListPartsRequest(bucketName, objectName, uploadId);
var listPartsResult = client.ListParts(listPartsRequest);

// Traverses all parts
var parts = listPartsResult.Parts;
foreach (var part : parts)
{
  Console.WriteLine( “PartNumber:” + part. PartNumber + “, ETag: ” + part. Etag +”,Size:” + part.Size);
```

```
}
```

NOTE:Under normal conditions, if a bucket contains more than 1000 multipart upload tasks, the first 1000 will be returned and the IsTruncated parameter in the returned results will be false. The returned NextPartNumberMarker can be used as the next start point to continue reading the data.If the user wishes to increase the number of upload tasks returned, he can modify the MaxParts parameter or use the PartNumberMarker parameters for segmented reading.

Anti-leech Settings

The OSS collects service fees based on use. To prevent users' data on OSS from being leeched, OSS supports anti-leech based on the field referer in HTTP header.

Setting the Referer White List

We can use the following code to set the Referer white list:

```
var client = new OssClient(endpoint, accessId, accessKey);

var refererList = new List<string>();
// Adds referer
refererList.Add(" http://www.aliyun.com");
refererList.Add(" http://www.*.com");
refererList.Add(" http://www.?.aliyuncs.com");
// Allows the referer field to be blank and sets the Bucket Referer List
var request = new SetBucketRefererRequest(bucketName, refererList);
request. AllowEmptyReferer = true;

client.setBucketReferer(bucketName, br);
```

The Referer parameter supports the wildcards "*" and "?". For detailed rule configuration, refer to the product documentation OSS Anti-leech

Retrieving the Referer White List

```
var rc = client.GetBucketReferer(bucketName);
Console.WriteLine("allow ? " + (rc.AllowEmptyReferer ? "yes" : "no"));
if (rc.RefererList.Referers != null)
{
for (var i = 0; i < rc.RefererList.Referers.Length; i++)
        Console.WriteLine(rc.RefererList.Referers[i]);
}
else
{
```

```
    Console.WriteLine("Empty Referer List");
}
```

Clearing the Referer White List

The Referer white list cannot be cleared directly. You can only reset it to overwrite the previous rules.

```
var client = new OssClient(endpoint, accessId, accessKey);
// Allows the referer field and the referer white list name to be blank by default
var request = new SetBucketRefererRequest(bucketName);
client.SetBucketReferer(request);
```

Lifecycle Management

OSS provides the object lifecycle management capability to manage objects for users. The user can configure the lifecycle of a bucket to define various rules for the bucket's objects. Currently, users can use rules to delete matched objects. Each rule is composed of the following parts:

- The object name prefix; this rule will only apply to objects with the matched prefix.
- Operation; the operation the user wishes to perform on the matched objects.
- Date or number of days; the user will execute the operation on the objects on the specified date or a specified number of days after the object's last modification time.

Setting Lifecycles

The lifecycle configuration rules are expressed by an xml segment.

```
<LifecycleConfiguration>
  <Rule>
    <ID>delete obsoleted files</ID>
    <Prefix>obsoleted/</Prefix>
    <Status>Enabled</Status>
    <Expiration>
      <Days>3</Days>
    </Expiration>
  </Rule>
  <Rule>
    <ID>delete temporary files</ID>
    <Prefix>temporary/</Prefix>
    <Status>Enabled</Status>
    <Expiration>
      <Date>2022-10-12T00:00:00.000Z</Date>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

A single lifecycle Config can contain up to 1000 rules.

Explanations of each field:

- The ID field is used to uniquely identify a rule (inclusion relations, such as abc and abcd, cannot exist between IDs).
- Prefix indicates the rules used for objects in the bucket with the specified prefix.
- Status indicates the status of this rule. The statuses are Enabled and Disabled, indicating if the rule is enabled or disabled.
- In the Expiration node, Days indicates that an object will be deleted a specified number of days after its last modification. Date indicates that objects will be deleted after the specified absolute time (the absolute time follows the ISO8601 format).

Using the following code, we can set the above lifecycle rules.

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

var setBucketLifecycleRequest = new SetBucketLifecycleRequest(bucketName);
LifecycleRule lcr1 = new LifecycleRule()
{
    ID = "delete obsoleted files",
    Prefix = "obsoleted/",
    Status = RuleStatus.Enabled,
    ExpriationDays = 3
};
LifecycleRule lcr2 = new LifecycleRule()
{
     ID = "delete temporary files",
     Prefix = "temporary/",
     Status = RuleStatus.Enabled,
     ExpirationTime = DateTime.Parse("2022-10-12T00:00:00.000Z")
};
setBucketLifecycleRequest.AddLifecycleRule(lcr1);
setBucketLifecycleRequest.AddLifecycleRule(lcr2);

client.SetBucketLifecycle(setBucketLifecycleRequest);
```

We can use the following code to retrieve the above lifecycle rules.

```
var rules = client.GetBucketLifecycle(bucketName);
foreach (var rule in rules)
{
     Console.WriteLine("ID: {0}", rule.ID);
     Console.WriteLine("Prefix: {0}", rule.Prefix);
     Console.WriteLine("Status: {0}", rule.Status);
     if (rule.ExpriationDays.HasValue)
          Console.WriteLine("ExpirationDays: {0}", rule.ExpriationDays);
     if (rule.ExpirationTime.HasValue)
          Console.WriteLine("ExpirationTime: {0}", FormatIso8601Date(rule.ExpirationTime.Value));
}
```

Using the following code, we can delete the lifecycle rules in a bucket.

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

var LifecycleRequest = new SetBucketLifecycleRequest(bucketName);
client.SetBucketLifecycle(LifecycleRequest);
```

Authorized Access

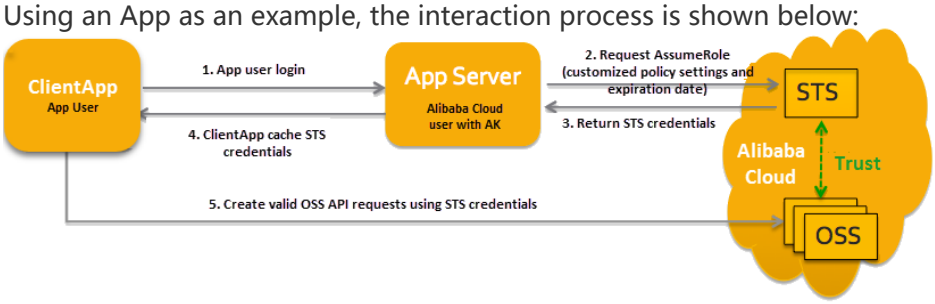
Using STS Service Temporary Authorization

Introduction

Through the AliCloud STS service, OSS can temporarily grant authorized access.AliCloud STS is a web service that provides a temporary access token to a cloud computing user. Using STS, you can grant access credentials to a third-party application or federated user (you can manage the user IDs) with customized permissions and validity periods. Third-party applications or federated users can use these access credentials to directly call the AliCloud product APIs or use the SDKs provided by AliCloud products to access the cloud product APIs.

- You do not need to expose you long-term key (AccessKey) to a third-party application and only need to generate an access token and send the access token to the third-party application. You can customize the access permission and validity of this token.
- You do not need to care about permission revocation issues. The access credential automatically becomes invalid when it expires.

Using an App as an example, the interaction process is shown below:



The solution is described in detail as follows:

1. Log in as the app user. App user IDs are managed by the client. Clients can customize the ID management system and may also use external Web accounts or OpenID. For each valid app user, the AppServer can precisely define the minimum access permission.
2. The AppServer requests a security token from the STS. Before calling STS, the AppServer needs to determine the minimum access permission (described in policy syntax) of app

users and the expiration time of the authorization.Then, the security token is obtained by calling the STS' AssumeRole interface.
3. The STS returns a valid access credential to the AppServer, where the access credential includes a security token, a temporary access key (AccessKeyId and AccessKeySecret), and the expiry time.
4. The AppServer returns the access credential to the ClientApp. The ClientApp caches this credential. When the credential becomes invalid, the ClientApp needs to request a new valid access credential from the AppServer. For example, if the access credential is valid for one hour, the ClientApp can request the AppServer to update the access credential every 30 minutes.
5. The ClientApp uses the access credential cached locally to request for AliCloud Service APIs. The ECS is aware of the STS access credential, relies on STS to verify the credential, and correctly responds to the user request.

The key is to obtain a valid access credential by simply calling the STS interface AssumeRole.The method can also be called by using the STS DSK. Clicking to View Details

Using STS Credentials to Construct Signed Requests

After obtaining the STS temporary credential, the user's client generates an OSSClient using the contained SecurityToken and temporary access key (AccessKeyId, AccessKeySecret). Using an object upload as an example:

```
String accessKeyId = "<accessKeyId>";
String accessKeySecret = "<accessKeySecret>";
String securityToken = "<securityToken>"
// Uses Hangzhou as an example
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";

var ossClient = new OssClient(endpoint, accessKeyId, accessKeySecret,securityToken);
```

Using URL Signature to Authorize Access

Generating a Signed URL

You can provide users with a temporary access URL by generating a signed URL. During URL generation, you can specify the URL expiration time to limit the duration of the user's access.

Generating a Signed URL

The code is as follows:

```
Svar req = new GeneratePresignedUriRequest(bucketName, key, SignHttpMethod.Get);
{
```

```
Expiration = new DateTime().AddHours(1)
}
var uri = client.GeneratePresignedUri(req);
```

Generated URLs use the GET access method by default. This way, users can directly use a browser to access the relevant content.

Generating Other HTTP Method URLs

For users to temporarily use other operations (e.g. uploading or deleting objects), you may have to sign a URL for another method. For example:

```
// Generates a PUT method URL
var req = new GeneratePresignedUriRequest(bucketName, key, SignHttpMethod.Put);
{
Expiration = new DateTime().AddHours(1),
ContentType =  "text/html"
}
var uri = client.GeneratePresignedUri(req);
```

Using Signed URLs to Send Requests

Currently, the .NET SDK supports the put object and get object URL signature requests.

Using the putobject URL Signature Method

```
var generatePresignedUriRequest = new GeneratePresignedUriRequest(bucketName, key, SignHttpMethod.Put);
var signedUrl = client.GeneratePresignedUri(generatePresignedUriRequest);
var result = client.PutObject(signedUrl, fileToUpload);
```

Cross-Origin Resource Sharing (CORS)

CORS allows Web applications to access resources in other origins. OSS provides an interface to allow developers to easily control cross-origin access permissions.

Setting CORS Rules

Using the setBucketCORS method, we can set a CORS rule for a specified bucket. If an original rule exists, it will be overwritten. Parameters for specific rules are generally set through the CORSRule class. The code is as follows:

```
// Initialize OSSClient
```

```
var client = new OssClient(endpoint, accessId, accessKey);

var req = new SetBucketCorsRequest(bucketName);
var r1 = new CORSRule();
//Specifies where cross-origin requests can originate from
r1.AddAllowedOrigin("http://www.a.com");
//Specifies the allowed cross-origin request methods (GET/PUT/DELETE/POST/HEAD)
r1.AddAllowedMethod("POST");
//Controls whether the headers specified by Access-Control-Request-Headers in the OPTIONS' prefetch command
are allowed.
r1.AddAllowedHeader("*");
//Specifies the response headers users are allowed to access from the application
r1.AddExposeHeader("x-oss-test");
req.AddCORSRule(r1);
client.SetBucketCors(req);
```

Here, you must particularly note the following:

- Each Bucket allows up to 10 rules.
- The AllowedOrigins and AllowedMethods each supports up to one "*" wildcard. "*" indicates that all origins or methods are allowed. However, AllowedHeaders and ExposeHeaders do not support wildcards.

Retrieving CORS Rules

We can refer to the bucket's CORS rules through the GetBucketCORSRules method. The code is as follows:

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

foreach (var rule in rules)
{
    Console.WriteLine( “AllowedOrigins:”  + rule. AllowedOrigins);
    Console.WriteLine( “AllowedMethods:”  + rule. AllowedMethods);
    Console.WriteLine( “AllowedHeaders:”  + rule. AllowedHeaders);
    Console.WriteLine( “ExposeHeaders:”  + rule. ExposeHeaders);
Console.WriteLine( “MaxAgeSeconds:”  + rule. MaxAgeSeconds);
}
```

Deleting CORS Rules

Use this to disable CORS for the specified bucket and clear all rules.

```
// Initialize OSSClient
var client = new OssClient(endpoint, accessId, accessKey);

// Clears the CORS rules in the bucket
client.DeleteBucketCors(bucketName);
```

Exceptions

The OSS .NET SDK has two exceptions: ClientException and OSSException. Both are derived, directly or indirectly, from RuntimeException.

ClientException

ClientException indicates an internal SDK exception, such as no set BucketName, cannot connect to the network, etc.

OSSException

OSSException indicates a server error, which is generated by parsing a server error message. OSSExceptions generally have the following components:

- Code: the error code OSS returns to users.
- Message: the detailed error message provided by OSS.
- RequestId: the UUID that uniquely identifies the request. When you cannot solve the problem, you can seek help from OSS development engineers by providing this RequestId.
- HostId: used to identify the accessed OSS cluster (currently unified as oss.aliyuncs.com)

The following are common OSS exceptions:

Error Code	Description
AccessDenied	Access denied
BucketAlreadyExists	The bucket already exists
BucketNotEmpty	The bucket is not empty
EntityTooLarge	The entity is too large
EntityTooSmall	The entity is too small
FileGroupTooLarge	The file group is too large
FilePartNotExist	A file part does not exist
FilePartStale	A file part has expired
InvalidArgument	Parameter format error
InvalidAccessKeyId	The Access Key ID does not exist
InvalidBucketName	The bucket name is invalid
InvalidDigest	The digest is invalid
InvalidObjectName	The object name is invalid

InvalidPart	A part is invalid
InvalidPartOrder	The part order is invalid
InvalidTargetBucketForLogging	The logging operation has an invalid target bucket
InternalError	Internal OSS error
MalformedXML	Illegal XML format
MethodNotAllowed	The method is not supported
MissingArgument	A parameter is missing
MissingContentLength	The content length is missing
NoSuchBucket	The bucket does not exist
NoSuchKey	The file does not exist
NoSuchUpload	The Multipart Upload ID does not exist
NotImplemented	The method cannot be processed
PreconditionFailed	Preprocessing error
RequestTimeTooSkewed	The request initiation time exceeds the server time by 15 minutes
RequestTimeout	Request timed out
SignatureDoesNotMatch	Signature error
TooManyBuckets	The user's bucket quantity exceeds the limit

PHP-SDK

OSS PHP SDK Documentation

Introduction

- Introduction
    - The Object Storage Service (OSS) is a massive, secure, cost-effective and highly reliable cloud storage service provided by AliCloud. Users can upload and download data anytime, anywhere and on any Internet device through a simple RESTful interface described herein. With the OSS, users can develop a diverse range of massive data–based services such as multimedia sharing websites, online

storage, personal and corporate data backups.

ChangeHistory

2015.7.1

- Added settings for response body conversion. The OSS currently supports xml, array, and json formats. XML is the default format
- Added the copy_upload_part method
- Added support for STS
- Changed the $options parameter location in the signature URL
- Fixed the read_dir looping problem *2015.3.30
- Added the referer and lifecycle interfaces. Added the content-md5 check option for upload by file and multipart upload.
- Added init_multipart_upload to directly obtain string type uploads
- Adjusted the return value of the batch_upload_file function from the original blank value to a boolean value; true indicates success and false indicates failure.

  Adjusted the tool function location in tsdk.class.php, placing it in util/oss_util.class.php. If you need to reference it, add OSSUtil:: and reference this file.

  Bug fixes:

- Fixed the problem in the Copy object process where you could not edit the header.
- Fixed the custom upload syntax error during upload part.
- Fixed the problem where the mimetype of office2007 files could not be set correctly during uploads.
- Fixed the problem where the system would time out and quit when it encountered an empty directory during the batch_upload_file operation.

Naming Rules

- Bucket naming rules
  - It can only contain lower-case letters, numbers, and hyphens (-)
  - It must start with a lower-case letter or number
  - The length must be 3-63 bytes
- Object naming rules
  - It uses UTF-8 encoding
  - The length must be 1-1023 bytes
  - It cannot start with "/" or "\"
  - It cannot contain "\r" or "\n" line breaks

Pre-dependency Check

- PHP extension library detection
  - Before use, please check if the curl, mbstring, SimpleXML, json, iconv, and other extension libraries are enabled. If not, please modify php.ini to enable the relevant extension libraries

Initializing Resources

Normal ALIOSS initialization method

- In normal conditions, initialization only requires you to provide your accessId, accessKey, endPoint, and other information. Moreover, you need to complete the required information

```
$access_id = "Please enter the accessId";
$access_key = "Please enter the accessKey";
$end_point = "The endpoint of the operation cluster";
$client = ALIOSS($access_id,$access_key,$end_point);

//Determines whether shown in domain name format. If true, the data is displayed in domain name format, e.g.
http://bucket.domain/object,
//If false, its format is http://domain/bucket/object
$client->set_enable_domain_style(true);
```

- Using STS to initialize ALIOSS
  - To use the STS service, mobile developers only need to call the STS API during initialization to generate a temporary accessId, accessKey, and securityToken.

```
$access_id = "Calls the STS interface to get a temporary access_id";
$access_key = "Calls the STS interface to get a temporary access_key";
$end_point = "The endpoint of the operation cluster";
$security_token = "Calls the STS interface to get a temporary security_token";
$client = ALIOSS($access_id,$access_key,$end_point,$security_token);
```

Bucket-related Operations

- Getting a bucket list
  - Sample Code

```
$options = null;
$response = $client->list_bucket($options);
```

- Parameter description

```
$options: optional
```

Response result

```
`
Get the result by converting the resulting response to array. The same below
  Array(
    [status] => 200
    [header] => Array(
      [date] => Wed, 01 Jul 2015 09:21:15 GMT
      [content-type] => application/xml
      [content-length] => 6266
      [connection] => close
      [server] => AliyunOSS
      [x-oss-request-id] => 5593B10B58DB3AB752154A62
    )
    [body] => Array(
      [ListAllMyBucketsResult] => Array(
        [Owner] => Array (
          [ID] => 128257
          [DisplayName] => 128257
        )
        [Buckets] => Array(
          [Bucket] => Array(
            [0] => Array(
              [Location] => oss-cn-hangzhou
               [Name] => 33331111
               [CreationDate] => 2014-08-27T03:04:20.000Z
             )
             [1] => Array (
               [Location] => oss-cn-qingdao
               [Name] => a-00000000000000000001
               [CreationDate] => 2015-05-22T05:30:40.000Z
             )

           )

         )

      )

    )

  )
`
```

- Creating a bucket

- Sample Code

```
$bucket_name = "bucket name";
```

```
$acl = ALIOSS::OSS_ACL_TYPE_PRIVATE;
$options = null;
$response = $client->create_bucket($bucket_name,$acl,$options);
```

- Parameter description

```
$bucket_name: a required parameter. It must comply with the bucket naming rules
$acl: a required parameter. It must be any of these values: private, public-read, or public-read-write, which are respectively mapped with the following constants
    ALIOSS::OSS_ACL_TYPE_PRIVATE,
    ALIOSS::OSS_ACL_TYPE_PUBLIC_READ,
    ALIOSS::OSS_ACL_TYPE_PUBLIC_READ_WRITE
$options: optional
```

- Response result

```
Get the result by converting the resulting response to array. The same below
Array(
    [status] => 200
    [header] => Array(
        [date] => Wed, 01 Jul 2015 09:55:18 GMT
        [content-length] => 0
        [connection] => close
        [server] => AliyunOSS
        [x-oss-request-id] => 5593B906031C87E546154CC1
    )

    [body] =>
)
```

- Deleting a bucket
    - Sample Code

```
$bucket_name = "bucket name";
$options = null;
$response = $client->delete_bucket($bucket_name,$options);
```

- Parameter description

```
$bucket_name: a required parameter. It must comply with the bucket naming rules
$options: optional
```

- Response result

```
Get the result by converting the resulting response to array. The same below
Array(
    [status] => 204
```

```
    [header] => Array(
        [date] => Wed, 01 Jul 2015 10:08:45 GMT
        [content-length] => 0
        [connection] => close
        [server] => AliyunOSS
        [x-oss-request-id] => 5593BC2D58DB3AB752155156
    )

    [body] =>
)
```

- Getting a bucket ACL
    - Sample Code

```
$bucket_name = "bucket name";
$options = null;
$response = $client->get_bucket_acl($bucket_name,$options);
```

- Parameter description

```
$bucket_name: a required parameter. It must comply with the bucket naming rules
$options: optional
```

- Response result

```
Get the result by converting the resulting response to array. The same below

Array(
    [status] => 200
    [header] => Array(
        [date] => Wed, 01 Jul 2015 10:17:41 GMT
        [content-type] => application/xml
        [content-length] => 239
        [connection] => close
        [server] => AliyunOSS
        [x-oss-request-id] => 5593BE45031C87E54615500F
    )

    [body] => Array(
        [AccessControlPolicy] => Array(
            [Owner] => Array(
                [ID] => 128257
                [DisplayName] => 128257
            )

            [AccessControlList] => Array(
                [Grant] => public-read
            )
        )
    )
)
```

- Setting the bucket ACL
  - Sample Code

```
$bucket_name = "bucket name";
$options = null;
$response = $client->set_bucket_acl($bucket_name,$options);
```

- Parameter description

```
$bucket_name: a required parameter. It must comply with the bucket naming rules
$options: optional
```

- Response result

```
Get the result by converting the resulting response to array. The same below
   Array(
      [status] => 200
      [header] => Array(
         [date] => Wed, 01 Jul 2015 11:08:31 GMT
         [content-length] => 0
         [connection] => close
         [server] => AliyunOSS
         [x-oss-request-id] => 5593CA2F031C87E5461557B5
      )

      [body] =>
   )
```

Object-related Operations

- Getting the object list
  - Sample Code

```
$bucket_name = 'bucket name';
$options = array(
    'delimiter' => '/',
    'prefix' => '',
    'max-keys' => 5,
    'marker' => '',
);

$response = $client->list_object($$bucket_name,$options);
```

- Parameter description

```
$bucket_name: a required parameter. It must comply with the bucket naming rules
```

$options: optional, as described below
Delimiter is a character used to group Object names. All objects whose names contain the specified prefix and that appear between the Delimiter characters for the first time are used as a group of elements:
CommonPrefixes.
Prefix requires the returned object key to be prefixed with prefix. Note that when querying using prefix, the returned keys will still contain prefix
max-keys limits the maximum number of objects returned for one request. If not specified, the default value is 100. The max-keys value cannot exceed 1,000.
Marker sets up the returned results to begin from the first entry after the Marker in alphabetical order.

- Response result

Get the result by converting the resulting response to array. The same below

```
Array(
    [status] => 200
    [header] => Array(
        [date] => Thu, 02 Jul 2015 07:59:32 GMT
        [content-type] => application/xml
        [content-length] => 1466
        [connection] => close
        [server] => AliyunOSS
        [x-oss-request-id] => 5594EF64031C87E546160F24
    )

    [body] => Array(
        [ListBucketResult] => Array(
            [Name] => common-bucket
            [Prefix] =>
            [Marker] =>
            [MaxKeys] => 5
            [Delimiter] => /
            [IsTruncated] => true
            [NextMarker] => metro_driver.dll
            [Contents] => Array(
                [0] => Array(
                    [Key] => chrome_elf.dll
                    [LastModified] => 2015-07-01T03:44:58.000Z
                    [ETag] => "78CE940FD1CCDF6F743EE1A9AED8AAD8"
                    [Type] => Normal
                    [Size] => 133960
                    [StorageClass] => Standard
                    [Owner] => Array(
                        [ID] => 128257
                        [DisplayName] => 128257
                    )
                )

                [1] => Array(
                    [Key] => delegate_execute.exe
                    [LastModified] => 2015-06-29T09:18:41.000Z
                    [ETag] => "37C49C4E0EC4E0D96B6EBBA2190E8824"
                    [Type] => Normal
                    [Size] => 692040
                    [StorageClass] => Standard
```

```
                [Owner] => Array(
                    [ID] => 128257
                    [DisplayName] => 128257
                )
            )
        }

        [CommonPrefixes] => Array(
            [0] => Array(
                [Prefix] => common-folder/
            )

            [1] => Array(
                [Prefix] => common-folder2/
            )
        )
    )
)
```

- Creating a simulated folder
    - Sample Code

```
$bucket_name = 'bucket name';
$dir_name = 'directory name';
$options = null;
$response  = $client->create_object_dir($bucket_name,$dir_name,$options);
```

- Parameter description

```
$bucket_name: a required parameter. Must comply with the bucket naming rules
$dir_name: a required parameter. Must comply with the object naming rules
$options: optional. You do not need to set a 'Header' header for simulated folders
```

- Response result

```
Get the result by converting the resulting response to array. The same below

Array(
    [status] => 200
    [header] => Array(
        [date] => Thu, 02 Jul 2015 08:10:16 GMT
        [content-length] => 0
        [connection] => close
        [etag] => "D41D8CD98F00B204E9800998ECF8427E"
        [server] => AliyunOSS
        [x-oss-request-id] => 5594F1E8031C87E5461610B9
    )

    [body] =>
)
```

- Uploading files (directly specified content)

• Sample Code

```
$bucket_name = 'bucket name';
$object_name = 'object name';
$content  = 'object content';
$options = array(
    'content' => $content,
    'length' => strlen($content),
    ALIOSS::OSS_HEADERS => array(
        'Expires' => 'Fri, 28 Feb 2012 05:38:42 GMT',
        'Cache-Control' => 'no-cache',
        'Content-Disposition' => 'attachment;filename=oss_download.log',
        'Content-Encoding' => 'utf-8',
        'Content-Language' => 'zh-CN',
        'x-oss-server-side-encryption' => 'AES256',
    ),
);
$response = $obj->upload_file_by_content($bucket_name,$object_name,$options);
```

- Parameter description

```
$bucket_name: a required parameter. Must comply with the bucket naming rules
$object_name: a required parameter. It must comply with the object naming rules
$options: a required parameter. It specifies the various information required for the upload. A detailed description is given below
    content: the content of the object to upload
    length: the size of the object to upload
    ALIOSS::OSS_HEADERS: optional parameter; if specified, it indicates meta information for this object and the following header information can be configured:
        Expires is the expiration time (in milliseconds)
        Cache-Control specifies the cache action of the web page when the object is downloaded
        Content-Disposition specifies the name of the object when downloaded
        Content-Encoding specifies the content encoding format when the object is downloaded
        Content-Language specifies the object language when downloaded
        x-oss-server-side-encryption specifies the server-side encryption algorithm used when the OSS creates an object
```

- Response result

```
Get the result by converting the resulting response to array. The same below

Array(
    [status] => 200
    [header] => Array(
        [date] => Thu, 02 Jul 2015 08:24:11 GMT
        [content-length] => 0
        [connection] => close
        [etag] => "9BA9EF6DDFBE14916FA2D3337B427774"
        [server] => AliyunOSS
        [x-oss-request-id] => 5594F52B031C87E5461612D1
```

```
        )

        [body] =>
    )
```

- Uploading files (with the file path specified)
    - Sample Code

```
    $bucket_name = 'bucket name';
    $object_name = 'object name';
    $file_path = "upload file path";
    $options = array(
        ALIOSS::OSS_HEADERS => array(
            'Expires' => 'Fri, 28 Feb 2012 05:38:42 GMT',
            'Cache-Control' => 'no-cache',
            'Content-Disposition' => 'attachment;filename=oss_download.gz',
            'Content-Encoding' => 'utf-8',
            'Content-Language' => 'zh-CN',
            'x-oss-server-side-encryption' => 'AES256',

        ),
    );
    $response = $obj->upload_file_by_file($bucket,$object,$file_path,$upload_file_options);
```

- Parameter description

```
    $bucket_name: a required parameter. Must comply with the bucket naming rules
    $object_name: a required parameter. It must comply with the object naming rules
    $file_path: a required parameter. It indicates the path of the file
    $options: a required parameter. It specifies the various information required for the upload. A detailed
description is given below
        content: the content of the object to upload
        length: the size of the object to upload
        ALIOSS::OSS_HEADERS: optional parameter; if specified, it indicates meta information for this object and the
following header information can be configured:
            Expires is the expiration time (in milliseconds)
            Cache-Control specifies the cache action of the web page when the object is downloaded
            Content-Disposition specifies the name of the object when downloaded
            Content-Encoding specifies the content encoding format when the object is downloaded
            Content-Language specifies the object language when downloaded
            x-oss-server-side-encryption specifies the server-side encryption algorithm used when the OSS creates an
object
```

- Response result

```
  Get the result by converting the resulting response to array. The same below

    Array(
        [status] => 200
        [header] => Array(
            [date] => Thu, 02 Jul 2015 08:41:10 GMT
```

```
            [content-length] => 0
            [connection] => close
            [etag] => "4B12FF064A3BBFE0AE5A1314E77FF0DF"
            [server] => AliyunOSS
            [x-oss-request-id] => 5594F91358DB3AB7521617CA
        )

        [body] =>
    )
```

- Copying objects
    - Sample Code

```
$from_bucket = 'copy from bucket';
$from_object = 'copy from object';
$to_bucket = 'copy to bucket';
$to_object = 'copy to object';
$options = array(
    ALIOSS::OSS_HEADERS => array(
        'x-oss-copy-source-if-match' => 'E024E425254F1EEDB237F69F854CE883',
        'x-oss-copy-source-if-none-match' => 'Thu, 18 Jun 2015 08:50:31 GMT',
        'x-oss-copy-source-if-unmodified-since' => 'Thu, 18 Jun 2015 08:50:31 GMT',
        'x-oss-copy-source-if-modified-since' => 'Thu, 18 Jun 2015 09:50:31 GMT',
        'x-oss-metadata-directive' => 'COPY',
        'x-oss-server-side-encryption' => 'AES256'
    )
);

$response = $obj->copy_object($from_bucket,$from_object,$to_bucket,$to_object,$options);
```

- Parameter description

```
    $from_bucket: a required parameter. It indicates the source bucket and must comply with the bucket naming
rules
    $from_object: required parameter, the source object; must comply with the object naming rules
    $to_bucket: required parameter, the destination bucket; must comply with the bucket naming rules
    $to_object: required parameter, the destination object; must comply with the object naming rules
    $options: optional parameter; if specified, it can define the ALIOSS::OSS_HEADERS header parameter, which
has several options shown below
        x-oss-copy-source-if-match: If the source object's ETAG value is the same as the ETag provided by the user,
a copy operation will be executed.
        Otherwise, the system returns the 412 HTTP error code (preprocessing failed)
        x-oss-copy-source-if-none-match: If the source object has not been modified after the time specified by the
user, the system performs a copy operation.
        Otherwise, the system returns the 412 HTTP error code (preprocessing failed)
        x-oss-copy-source-if-unmodified-since: If the time in the parameter is the same as or later than the file's
actual modification time, the file is transmitted normally
        and the system returns 200 OK. Otherwise, the system throws the 412 precondition failed exception
        x-oss-copy-source-if-modified-since: If the source object has been modified after the time specified by the
user, the system performs a copy operation.
        Otherwise, the system returns the 412 HTTP error code (preprocessing failed)
        x-oss-metadata-directive: Values: COPY and REPLACE. If the parameter is set to COPY, the new object's meta
is all copied from the source object.
```

If the parameter is set to REPLACE, the source object's meta is ignored and the meta values specified by the user in this request are used. Other values may cause the system to return an 400 HTTP error code.

Note that when the value is COPY, the source object's x-oss-server-side-encryption meta value cannot be copied. The default value is COPY

x-oss-server-side-encryption specifies the server entropy encryption algorithm used when the OSS creates the destination object. The system currently only supports AES256

- Response result

Get the result by converting the resulting response to array. The same below

```
Array(
    [status] => 200
    [header] => Array(
        [content-type] => application/xml
        [content-length] => 184
        [connection] => close
        [date] => Thu, 02 Jul 2015 09:26:28 GMT
        [etag] => "E024E425254F1EEDB237F69F854CE883"
        [server] => AliyunOSS
        [x-oss-request-id] => 559503C458DB3AB752161E83
        [x-oss-server-side-encryption] => AES256
    )

    [body] => Array(
        [CopyObjectResult] => Array(
            [LastModified] => 2015-07-02T09:26:28.000Z
            [ETag] => "E024E425254F1EEDB237F69F854CE883"
        )
    )
)
```

- Getting Object MetaData
    - Sample Code

```
$bucket_name = 'bucket name';
$object_name = 'object name';
$options = null;
$response = $client->get_object_meta($bucket_name,$object_name,$options);
```

- Parameter description

```
$bucket_name: a required parameter. Must comply with the bucket naming rules
$object_name: a required parameter. It must comply with the object naming rules
$options: optional
```

- Response result

```
Array(
```

```
    [status] => 200
    [header] => Array(
        [date] => Thu, 02 Jul 2015 09:03:40 GMT
        [content-type] => plain/text
        [content-length] => 10
        [connection] => close
        [accept-ranges] => bytes
        [cache-control] => no-cache
        [content-disposition] => attachment;filename=oss_download.log
        [content-encoding] => utf-8
        [content-language] => zh-CN
        [etag] => "9BA9EF6DDFBE14916FA2D3337B427774"
        [expires] => Fri, 28 Feb 2012 05:38:42 GMT
        [last-modified] => Thu, 02 Jul 2015 08:38:10 GMT
        [server] => AliyunOSS
        [x-oss-object-type] => Normal
        [x-oss-request-id] => 5594FE6C031C87E5461618B6
        [x-oss-server-side-encryption] => AES256
    )

    [body] =>
)
```

- Deleting a single object
    - Sample Code

```
$bucket_name = 'bucket name';
$object_name = 'object name';
$options = null;
$response = $client->delete_object($bucket_name,$object_name,$options);
```

- Parameter description

```
$bucket_name: a required parameter. Must comply with the bucket naming rules
$object_name: a required parameter. It must comply with the object naming rules
$options: optional
```

- Response result

```
Get the result by converting the resulting response to array. The same below

Array(
    [status] => 204
    [header] => Array(
        [content-length] => 0
        [connection] => close
        [date] => Thu, 02 Jul 2015 10:01:00 GMT
        [server] => AliyunOSS
        [x-oss-request-id] => 55950BDC58DB3AB75216239D
    )
```

```
    [body] =>
)
```

- Deleting multiple objects
  - Sample Code

```
$bucket_name = 'bucket name';
$objects = array(
    'delete object 1',
    'delete object 2',
    ...
);

$options = array(
    'quiet' => false,
);

$response = $client->delete_objects($bucket_name,$objects,$options);
```

- Parameter description

```
$bucket_name: a required parameter. Must comply with the bucket naming rules
$objects: a required parameter. The object in this parameter must comply with the object naming rules
$options: an optional parameter. You can select the delete mode based on the actual situation. The quite parameter has two values, true|false.
    true: the body of the message returned by OSS only contains results for objects which encountered an error in the delete process. If all objects are deleted successfully, there is no message body.
    false: the body of the message returned by OSS contains results for each object deleted
```

- Response result

```
Get the result by converting the resulting response to array. The same below

Array(
[status] => 200
[header] => Array(
    [content-type] => application/xml
    [content-length] => 188
    [connection] => close
    [date] => Thu, 02 Jul 2015 10:06:00 GMT
    [server] => AliyunOSS
    [x-oss-request-id] => 55950D0858DB3AB752162459
)

[body] => Array(
    [DeleteResult] => Array(
        [Deleted] => Array(
            [0] => Array(
                [Key] => delegate_execute.exe
            )
```

```
            [1] => Array(
                [Key] => metro_driver.dll
            )
        )
    )
)
```

- Downloading objects
    - Sample Code

```
$bucket_name = 'download bucket';
$object_name = 'download object';

$options = array(
    ALIOSS::OSS_FILE_DOWNLOAD => "download path",
    ALIOSS::OSS_RANGE  => '0-1',
);

$response = $client>get_object($bucket_name,$object_name,$options);
```

- Parameter description

```
$bucket_name: a required parameter. Must comply with the bucket naming rules
$object_name: a required parameter. It must comply with the object naming rules
$options: a required parameter. This parameter must provide the ALIOSS::OSS_FILE_DOWNLOAD and
ALIOSS::OSS_RANGE options based on the actual situation.
Otherwise, all content is downloaded by default
```

- Response result

```
Get the result by converting the resulting response to array. The same below

Array(
    [status] => 206
    [header] => Array(
    )

    [body] =>
)
```

MultipartUpload-related Operations

- Initializing multipartUpload
    - Sample Code

```
$bucket_name = 'bucket name';
$object_name = 'object name';
```

```
$options = array(
    ALIOSS::OSS_HEADERS => array(
        'Expires' => 'Fri, 28 Feb 2012 05:38:42 GMT',
        'Cache-Control' => 'no-cache',
        'Content-Disposition' => 'attachment;filename=oss_download.log',
        'Content-Encoding' => 'utf-8',
        'Content-Type' => 'plain/text',
        'Content-Language' => 'zh-CN',
        'x-oss-server-side-encryption' => 'AES256',
    ),
);
$response = $client->initiate_multipart_upload($bucket_name,$object_name,$options);
```

- Parameter description

```
$bucket_name: a required parameter. Must comply with the bucket naming rules
$object_name: a required parameter. It must comply with the object naming rules
$options: a required parameter. It specifies the various information required for the upload. A detailed description is given below
    ALIOSS::OSS_HEADERS: optional parameter; if specified, it indicates meta information for this object and the following header information can be configured:
        Expires is the expiration time (in milliseconds)
        Cache-Control specifies the cache action of the web page when the object is downloaded
        Content-Disposition specifies the name of the object when downloaded
        Content-Encoding specifies the content encoding format when the object is downloaded
        'Content-Type' => 'plain/text' specifies the MIME type during object response
        Content-Language specifies the object language when downloaded
        x-oss-server-side-encryption specifies the server-side encryption algorithm used when the OSS creates an object
```

- Response result

Get the result by converting the resulting response to array. The same below

```
Array(
    [status] => 200
    [header] => Array(
        [content-type] => application/xml
        [content-length] => 234
        [connection] => close
        [date] => Thu, 02 Jul 2015 11:35:36 GMT
        [server] => AliyunOSS
        [x-oss-request-id] => 5595220858DB3AB7521631B1
        [x-oss-server-side-encryption] => AES256
    )

    [body] => Array(
        [InitiateMultipartUploadResult] => Array(
            [Bucket] => common-bucket
            [Key] => multipart-upload-1435836936
            [UploadId] => 154A34BD1FE24A90A025EB800AA392CC
        )
    )
```

```
)
```

- Uploading parts
  - Sample Code

```
$bucket_name = 'bucket name';
$object_name = 'object name';
$upload_id = 'upload id';
$options = array(
    'fileUpload' => 'upload path',
    'partNumber' => 1,
    'seekTo' => 1,
    'length' => 5242880,
);

$response = $client->upload_part($bucket_name,$object_name, $upload_id, $options);
```

- Parameter description

```
$bucket_name: a required parameter. Must comply with the bucket naming rules
$object_name: a required parameter. It must comply with the object naming rules
$upload_id: a required parameter. Upload parts correspond to multipart uploads IDs
$options: a required parameter. It specifies the various information required for the upload. A detailed
description is given below
    fileUpload: the path of the file to upload
    partNumber: the serial number of the part to upload
    seekTo: The byte to start with when accessing the upload file
    length: the slice size
```

- Response result

```
Get the result by converting the resulting response to array. The same below

Array(
    [status] => 200
    [header] => Array(
        [content-length] => 0
        [connection] => close
        [date] => Thu, 02 Jul 2015 11:35:36 GMT
        [etag] => "3AE3AD480200A26738F10CBF2FFBE8B6"
        [server] => AliyunOSS
        [x-oss-request-id] => 5595220858DB3AB7521631B3
        [x-oss-server-side-encryption] => AES256
    )

    [body] =>
)
```

- Copying upload parts
  - Sample Code

```
$from_bucket = 'copy from bucket';
$from_object = 'copy from object';
$to_bucket = 'copy to bucket';
$to_object = 'copy to object';
$part_number = 1;
$upload_id = 'copy to upload id';
$options = array(
    'start' => 0,
    'end' => 25000032,
);

$response = $client-
>copy_upload_part($from_bucket,$from_object,$to_bucket,$to_object,$part_number,$upload_id,$options);
```

- Parameter description

```
$from_bucket: a required parameter. It indicates the source bucket and must comply with the bucket naming
rules
$from_object: required parameter, the source object; must comply with the object naming rules
$to_bucket: required parameter, the destination bucket; must comply with the bucket naming rules
$to_object: required parameter, the destination object; must comply with the object naming rules
$part_number: required parameter; range: 1-10,000
$upload_id: required parameter; the uploadid returned by initializing multipartupload
$options: an optional parameter. It provides isFullCopy, startRange, endRange,
    ALIOSS::OSS_HEADERS, and other parameters. In ALIOSS::OSS_HEADERS, the parameters that can be set are
as follows:
        x-oss-copy-source-if-match: If the source object's ETAG value is the same as the ETag provided by the
user, a copy operation will be executed.
        Otherwise, the system returns the 412 HTTP error code (preprocessing failed)
        x-oss-copy-source-if-none-match: If the source object has not been modified after the time specified by
the user, the system performs a copy operation.
        Otherwise, the system returns the 412 HTTP error code (preprocessing failed)
        x-oss-copy-source-if-unmodified-since: If the time in the parameter is the same as or later than the file's
actual modification time, the file is transmitted normally
        and the system returns 200 OK. Otherwise, the system throws the 412 precondition failed exception
        x-oss-copy-source-if-modified-since: If the source object has been modified after the time specified by
the user, the system performs a copy operation.
        Otherwise, the system returns the 412 HTTP error code (preprocessing failed)
    isFullCopy: Whether or not full copy is enabled. If the parameter is set to true, you do not need to set
startRange and endRange
    startRange: If isFullCopy is false, this parameter becomes valid and specifies the start location for copying
the source object
    endRange: If isFullCopy is false, this parameter becomes valid and specifies the end location for copying the
source object
```

Response result

Get the result by converting the resulting response to array. The same below

```
Array
(
```

```
    [success] => 1
    [status] => 200
    [header] => Array
        (
            [date] => Thu, 06 Aug 2015 18:13:59 GMT
            [content-type] => application/xml
            [content-length] => 180
            [connection] => keep-alive
            [content-range] => bytes 11304368-11534335/11534336
            [etag] => "E95C28888F15B92B9C49C9ECEC53C958"
            [server] => AliyunOSS
            [x-oss-bucket-version] => 1438864637
            [x-oss-request-id] => 55C3A3E79646C3C03F40EA5E
        )

    [body] => Array
        (
            [CopyPartResult] => Array
                (
                    [LastModified] => 2015-08-06T18:13:59.000Z
                    [ETag] => "E95C28888F15B92B9C49C9ECEC53C958"
                )

        )

)
```

- Getting part lists
    - Sample Code

```
$bucket_name = 'bucket name';
$object_name = 'object name';
$upload_id = 'upload id';
$options = null;
$response = $client->list_parts($bucket_name,$object_name, $upload_id,$options);
```

- Parameter description

```
$bucket_name: a required parameter. Must comply with the bucket naming rules
$object_name: a required parameter. It must comply with the object naming rules
$upload_id: a required parameter. Upload parts correspond to multipart uploads IDs
$options: optional
```

- Response result

```
Get the result by converting the resulting response to array. The same below

Array(
    [status] => 200
    [header] => Array(
```

```
            [content-type] => application/xml
            [content-length] => 584
            [connection] => close
            [date] => Thu, 02 Jul 2015 11:35:40 GMT
            [server] => AliyunOSS
            [x-oss-request-id] => 5595220C031C87E546162F44
        )

        [body] => Array(
            [ListPartsResult] => Array(
                [Bucket] => common-bucket
                [Key] => multipart-upload-1435836813
                [UploadId] => B4D4B89F8B064A3D835D83D7805B49F3
                [StorageClass] => Standard
                [PartNumberMarker] => 0
                [NextPartNumberMarker] => 1
                [MaxParts] => 1000
                [IsTruncated] => false
                [Part] => Array(
                    [PartNumber] => 1
                    [LastModified] => 2015-07-02T11:35:40.000Z
                    [ETag] => "3AE3AD480200A26738F10CBF2FFBE8B6"
                    [Size] => 5242880
                )
            )
        )
    )
```

- Getting multipartUpload lists
    - Sample Code

```
$bucket_name = 'bucket name';
$options = array(
    'delimiter' => '/',
    'max-uploads' => 2,
    'key-marker' => '',
    'prefix' => '',
    'upload-id-marker' => ''
);
$response = $client->list_multipart_uploads($bucket_name,$options);
```

- Parameter description

```
$bucket_name: a required parameter. Must comply with the bucket naming rules
$options: an optional parameter. It provides the following parameters
    Delimiter is a character used to group Object names. All objects whose names contain the specified prefix and that appear between the Delimiter characters for the first time are used as a group of elements:
    CommonPrefixes. max-uploads specifies the maximum number of Multipart Upload events returned for one request. If not specified, the default value is 1,000. The max-keys value cannot exceed 1,000.
    key-marker is used with the upload-id-marker parameter to specify the starting position of the returned result. lIf the upload-id-marker parameter is not specified, the query result contains:
    Multipart events in which the lexicographic orders of all object names are greater than the value of the key-marker parameter. l If the upload-id-marker parameter is specified, the query result contains:
```

Multipart events in which the lexicographic orders of all object names are greater than the value of the key-marker parameter and Multipart Upload events in which the object name is the same as the value of the key-marker parameter but the Upload ID is greater than the value of the upload-id-marker parameter

Prefix requires the returned object key to be prefixed with prefix. Note that the keys returned from queries using a prefix will still contain the prefix

upload-id-marker is used with the key-marker parameter to specify the starting position of the returned result. l If the key-marker parameter is not specified, the OSS ignores the upload-id-marker parameter.

If the key-marker parameter is specified, the query result contains: Multipart events in which the lexicographic orders of all object names are greater than the value of the key-marker parameter and Multipart Upload events in which the object name is the same as the value of the key-marker parameter but the Upload ID is greater than the value of the upload-id-marker parameter.

- Response result

Get the result by converting the resulting response to array. The same below

```
Array(
    [status] => 200
    [header] => Array(
        [content-type] => application/xml
        [content-length] => 876
        [connection] => close
        [date] => Thu, 02 Jul 2015 12:01:50 GMT
        [server] => AliyunOSS
        [x-oss-request-id] => 5595282E031C87E546163301
    )

    [body] => Array(
        [ListMultipartUploadsResult] => Array(
            [Bucket] => common-bucket
            [KeyMarker] =>
            [UploadIdMarker] =>
            [NextKeyMarker] => multipart-upload-1435835648
            [NextUploadIdMarker] => 5C79DDEC71DE478AA4AD9E9AA8BFE6DE
            [Delimiter] => /
            [Prefix] =>
            [MaxUploads] => 2
            [IsTruncated] => true
            [Upload] => Array(
                [0] => Array(
                    [Key] => multipart-upload-1435835395
                    [UploadId] => 799C914C0EC3448BAC126849A1B1D6D0
                    [StorageClass] => Standard
                    [Initiated] => 2015-07-02T11:09:55.000Z
                )

                [1] => Array(
                    [Key] => multipart-upload-1435835648
                    [UploadId] => 5C79DDEC71DE478AA4AD9E9AA8BFE6DE
                    [StorageClass] => Standard
                    [Initiated] => 2015-07-02T11:14:08.000Z
                )
            )
        )
    )
```

```
)
```

- Terminating multipartUpload
    - Sample Code

```
$bucket_name = 'bucket name';
$object_name = 'object name';
$upload_id = 'upload id';
$options = null;
$response = $client->abort_multipart_upload($bucket_name,$object_name,$upload_id,$options);
```

- Parameter description

```
$bucket_name: a required parameter. Must comply with the bucket naming rules
$object_name: a required parameter. It must comply with the object naming rules
$upload_id: a required parameter. Upload parts correspond to multipart uploads IDs
$options: optional
```

- Response result

```
Get the result by converting the resulting response to array. The same below

Array(
    [status] => 204
    [header] => Array(
        [content-length] => 0
        [connection] => close
        [date] => Thu, 02 Jul 2015 11:53:52 GMT
        [server] => AliyunOSS
        [x-oss-request-id] => 55952650031C87E5461631E7
    )

    [body] =>
)
```

- Completing multipartUpload
    - Sample Code

```
$bucket_name = 'bucket name';
$object_name = 'object name';
$upload_id = 'upload id';

$upload_parts = array();
$upload_parts[] = array(
    'PartNumber' => 1,
    'ETag' => '3AE3AD480200A26738F10CBF2FFBE8B6'
);
$options = null;
$response = $client-
```

```
>complete_multipart_upload($bucket_name,$object_name,$upload_id,$upload_parts,$options);
```

- Parameter description

```
$bucket_name: a required parameter. Must comply with the bucket naming rules
$object_name: a required parameter. It must comply with the object naming rules
$upload_id: a required parameter. Upload parts correspond to multipart uploads IDs
$upload_parts: an array that contains the parts. It must contain the PartNumber and Etag
$options: optional
```

- Response result

```
Get the result by converting the resulting response to array. The same below

Array(
    [status] => 200
    [header] => Array(
        [content-type] => application/xml
        [content-length] => 331
        [connection] => close
        [date] => Thu, 02 Jul 2015 11:35:40 GMT
        [etag] => "003B6AEB546001A97D838E411025239A-1"
        [server] => AliyunOSS
        [x-oss-request-id] => 5595220C031C87E546162F45
        [x-oss-server-side-encryption] => AES256
    )

    [body] => Array(
        [CompleteMultipartUploadResult] => Array(
            [Location] => http://common-bucket.oss-cn-shanghai.aliyuncs.com/multipart-upload-1435836813
            [Bucket] => common-bucket
            [Key] => multipart-upload-1435836813
            [ETag] => "003B6AEB546001A97D838E411025239A-1"
        )
    )
)
```

Lifecycle Management

- Creating lifecycle rules
    - Sample Code

```
$bucket_name = 'bucket name';
$lifecycle = "
   <LifecycleConfiguration>
     <Rule>
       <ID>DaysRule</ID>
       <Prefix>days/</Prefix>
       <Status>Enabled</Status>
```

```
            <Expiration>
              <Days>1</Days>
            </Expiration>
          </Rule>
        </LifecycleConfiguration>" ;
    $options = null;
    $response = $client->set_bucket_lifecycle($bucket_name,$lifecycle,$options);
```

- Parameter description

```
    $bucket_name: a required parameter. Must comply with the bucket naming rules
    $lifecycle: a required parameter that defines the lifecycle rule. For details about the xml elements, please refer to the OSS API documentation:
    http://docs.aliyun.com/?spm=5176.383663.9.2.1hkILe#/pub/oss/api-reference/bucket&PutBucketLifecycle
    $options: optional
```

- Response result

```
  Get the result by converting the resulting response to array. The same below

    Array(
        [status] => 200
        [header] => Array(
            [date] => Thu, 02 Jul 2015 06:32:57 GMT
            [content-length] => 0
            [connection] => close
            [server] => AliyunOSS
            [x-oss-request-id] => 5594DB19031C87E5461601D1
        )

        [body] =>
    )
```

- Getting lifecycle rules
  - Sample Code

```
    $bucket_name = "bucket name";
    $options = null;
    $response = $client->get_bucket_lifecycle($bucket_name,$options);
```

- Parameter description

```
    $bucket_name: a required parameter. Must comply with the bucket naming rules
    $options: optional
```

- Response result

Get the result by converting the resulting response to array. The same below

```
Array(
    [status] => 200
    [header] => Array(
        [date] => Thu, 02 Jul 2015 06:32:57 GMT
        [content-type] => application/xml
        [content-length] => 243
        [connection] => close
        [server] => AliyunOSS
        [x-oss-request-id] => 5594DB1958DB3AB75216045C
    )

    [body] => Array(
        [LifecycleConfiguration] => Array(
            [Rule] => Array(
                [ID] => DaysRule
                [Prefix] => days/
                [Status] => Enabled
                [Expiration] => Array(
                    [Days] => 1
                )
            )
        )
    )
)
```

- Deleting lifecycle rules
    - Sample Code

```
$bucket_name = "bucket name";
$options = null;
$response = $client->delete_bucket_lifecycle($bucket_name,$options);
```

- Parameter description

```
$bucket_name: a required parameter. Must comply with the bucket naming rules
$options: optional
```

- Response result

Get the result by converting the resulting response to array. The same below

```
Array(
    [status] => 204
    [header] => Array(
        [date] => Thu, 02 Jul 2015 06:32:58 GMT
        [content-length] => 0
        [connection] => close
        [server] => AliyunOSS
        [x-oss-request-id] => 5594DB1A58DB3AB75216045D
```

```
        )

        [body] =>
    )
```

Cross-origin Resource Sharing (CORS)

- Creating CORS rules
    - Sample Code

```
$bucket_name = 'bucket name';

$cors_rule = array();

$cors_rule[ALIOSS::OSS_CORS_ALLOWED_HEADER]=array("x-oss-test");
$cors_rule[ALIOSS::OSS_CORS_ALLOWED_METHOD]=array("GET");
$cors_rule[ALIOSS::OSS_CORS_ALLOWED_ORIGIN]=array("http://www.b.com");
$cors_rule[ALIOSS::OSS_CORS_EXPOSE_HEADER]=array("x-oss-test1");
$cors_rule[ALIOSS::OSS_CORS_MAX_AGE_SECONDS] = 10;

$cors_rules=array($cors_rule);

$options = null;
$response = $obj->set_bucket_cors($bucket_name, $cors_rules,$options);
```

- Parameter description

```
$bucket_name: a required parameter. Must comply with the bucket naming rules
$cors_rules: Defines an array of CORS rules. Each rule must contain the following elements:
    ALIOSS::OSS_CORS_ALLOWED_ORIGIN: a required parameter, which specifies the allowed origins of cross-origin requests. Each rule may contain no more than one "*" symbol
    ALIOSS::OSS_CORS_ALLOWED_METHOD: a required parameter, which specifies the allowed cross-origin request methods. Select one or more from GET, PUT, POST, DELETE, and HEAD
    ALIOSS::OSS_CORS_ALLOWED_HEADER: optional. The parameter determines whether the headers specified by Access-Control-Request-Headers in the OPTIONS' prefetch command are allowed.
    Each header specified by Access-Control-Request-Headers must match a value in AllowedHeader. Each rule supports no more than one wildcard "*"
    ALIOSS::OSS_CORS_EXPOSE_HEADER: optional. The parameter indicates the response headers users are allowed to access from an application (e.g.: one Javascript XMLHttpRequest object) )
    The wildcard "*" is not allowed.
    ALIOSS::OSS_CORS_MAX_AGE_SECONDS: optional. The parameter specifies the cache time (in seconds) for the return results of browser prefetch (OPTIONS) requests to a specific resource.
    One CORSRule allows no more than one such parameter.

$options: optional
```

- Response result

```
Get the result by converting the resulting response to array. The same below
```

```
Array(
    [status] => 200
    [header] => Array(
        [date] => Thu, 02 Jul 2015 07:03:29 GMT
        [content-length] => 0
        [connection] => close
        [server] => AliyunOSS
        [x-oss-request-id] => 5594E241031C87E546160697
    )

    [body] =>
)
```

- Getting CORS rules
    - Sample Code

```
$bucket = 'bucket name';
$options = null;
$response = $client->get_bucket_cors($bucket_name,$options);
```

- Parameter description

```
$bucket_name: a required parameter. It must comply with the bucket naming rules
$options: optional
```

- Response result

```
Get the result by converting the resulting response to array. The same below

Array(
    [status] => 200
    [header] => Array(
        [date] => Thu, 02 Jul 2015 07:03:39 GMT
        [content-type] => application/xml
        [content-length] => 327
        [connection] => close
        [server] => AliyunOSS
        [x-oss-request-id] => 5594E24B58DB3AB752160920
    )

    [body] => Array(
        [CORSConfiguration] => Array(
            [CORSRule] => Array(
                [AllowedOrigin] => http://www.b.com
                [AllowedMethod] => GET
                [AllowedHeader] => x-oss-test
                [ExposeHeader] => x-oss-test1
                [MaxAgeSeconds] => 10
            )
        )
    )
```

```
)
```

- Determining whether or not the cross-origin request is allowed
  - Sample Code

```
$bucket_name = 'bucket name';
$object_name ='object name';
$origin = 'http://www.b.com';
$request_method = ALIOSS::OSS_HTTP_GET;
$request_headers = 'x-oss-test';
$options = null;

$response = $obj->options_object($bucket_name, $object_name, $origin, $request_method,
$request_headers,$options);
```

- Parameter description

```
$bucket_name: a required parameter. It must comply with the bucket naming rules
$object_name: a required parameter. It must comply with the object naming rules
$origin: a required parameter. It indicates the origin of a request, used to identify the cross-origin request
$request_method: a required parameter. It indicates the methods to be used in the actual request
$request_headers: a required parameter. It indicates the headers, except simple headers, for use in actual
requests
$options: optional
```

- Response result

```
Get the result by converting the resulting response to array. The same below

Array(
    [status] => 200
    [header] => Array(
        [date] => Thu, 02 Jul 2015 07:03:39 GMT
        [content-length] => 0
        [connection] => close
        [server] => AliyunOSS
        [x-oss-request-id] => 5594E24B031C87E5461606A1
    )

    [body] =>
)
```

- Deleting CORS rules
  - Sample Code

```
$bucket_name = "bucket name";
$options = null;
$response = $client->delete_bucket_cors($bucket_name,$options);
```

- Parameter description

```
$bucket_name: a required parameter. It must comply with the bucket naming rules
$options: optional
```

- Response result

```
Get the result by converting the resulting response to array. The same below

Array
(
    [status] => 204
    [header] => Array(
        [date] => Thu, 02 Jul 2015 07:03:39 GMT
        [content-length] => 0
        [connection] => close
        [server] => AliyunOSS
        [x-oss-request-id] => 5594E24B031C87E5461606A2
    )

    [body] =>
)
```

Static Website Hosting

- Setting websites
    - Sample Code

```
$bucket_name = 'bucket name';
$index_document = 'index.html';
$error_document = 'error.html';
$options = null;
$response = $client->set_bucket_website($bucket_name,$index_document,$error_document,$options);
```

- Parameter description

```
$bucket_name: a required parameter. It must comply with the bucket naming rules
$index_document: a required parameter. When the website function is enabled, the index_document parameter must be set
$error_document: an optional parameter. When the website function is enabled, you can choose whether to set the error_document
$options: optional
```

- Response result

```
Get the result by converting the resulting response to array. The same below
```

```
Array(
    [status] => 200
    [header] => Array(
        [date] => Thu, 02 Jul 2015 02:39:23 GMT
        [content-length] => 0
        [connection] => close
        [server] => AliyunOSS
        [x-oss-request-id] => 5594A45B58DB3AB75215E239
    )

    [body] =>
)
```

- Getting website settings
    - Sample Code

```
$bucket_name = "bucket name";
$options = null;
$response = $client->get_bucket_website($bucket_name,$options);
```

- Parameter description

```
$bucket_name: a required parameter. It must comply with the bucket naming rules
$options: optional
```

- Response result

```
Get the result by converting the resulting response to array. The same below

Array(
    [status] => 200
    [header] => Array(
        [date] => Thu, 02 Jul 2015 02:39:24 GMT
        [content-type] => application/xml
        [content-length] => 218
        [connection] => close
        [server] => AliyunOSS
        [x-oss-request-id] => 5594A45C031C87E54615DF98
    )

    [body] => Array(
        [WebsiteConfiguration] => Array(
            [IndexDocument] => Array(
                [Suffix] => index.html
            )

            [ErrorDocument] => Array(
                [Key] => error.html
            )
        )
    )
```

```
)
```

- Deleting websites
    - Sample Code

```
$bucket_name = "bucket name";
$options = null;
$response = $client->delete_bucket_website($bucket_name,$options);
```

- Parameter description

```
$bucket_name: a required parameter. It must comply with the bucket naming rules
$options: optional
```

- Response result

```
Get the result by converting the resulting response to array. The same below

Array(
    [status] => 204
    [header] => Array(
        [date] => Thu, 02 Jul 2015 02:39:24 GMT
        [content-length] => 0
        [connection] => close
        [server] => AliyunOSS
        [x-oss-request-id] => 5594A45C031C87E54615DF99
    )
    [body] =>
)
```

Logging

- Setting logging
    - Sample Code

```
$bucket_name = "bucket name";
$target_bucket_name = "logging target bucket";
$target_prefix = "logging file prefix";
$options = null;
$response = $client->set_bucket_logging($bucket_name,$target_bucket_name,$target_prefix,$options);
```

- Parameter description

```
$bucket_name: a required parameter. It must comply with the bucket naming rules and must be an existing
bucket of the owner
```

```
    $target_bucket_name: a required parameter. It indicates the bucket in which to save the logs and must be in the same cluster as the bucket to be logged
    $target_prefix: an optional parameter. If this parameter is specified, the log file name will be $target_prefix + OSS log naming rules
    $options: optional
```

- Response result

```
Get the result by converting the resulting response to array. The same below

    Array(
        [status] => 200
        [header] => Array(
            [date] => Thu, 02 Jul 2015 01:59:06 GMT
            [content-length] => 0
            [connection] => close
            [server] => AliyunOSS
            [x-oss-request-id] => 55949AEA031C87E54615D996
        )

        [body] =>
    )
```

- Getting logging settings
    - Sample Code

```
    $bucket_name = "bucket name";
    $options = null;
    $response = $client->get_bucket_logging($bucket_name,$options);
```

- Parameter description

```
    $bucket_name: a required parameter. Must comply with the bucket naming rules
    $options: optional
```

- Response result

```
Get the result by converting the resulting response to array. The same below

    Array(
        [status] => 200
        [header] => Array(
            [date] => Thu, 02 Jul 2015 02:14:09 GMT
            [content-length] => 235
            [connection] => close
            [server] => AliyunOSS
            [x-oss-request-id] => 55949E7158DB3AB75215DE78
        )
```

```
    [body] => Array(
        [BucketLoggingStatus] => Array(
            [LoggingEnabled] => Array(
                [TargetBucket] => a-00000000000000000003
                [TargetPrefix] => common-bucket-logging-
            )
        )
    )
)
```

- Deleting logging
    - Sample Code

```
$bucket_name = "bucket name";
$options = null;
$response = $client->get_bucket_logging($bucket_name,$options);
```

- Parameter description

```
$bucket_name: a required parameter. Must comply with the bucket naming rules
$options: optional
```

- Response result

```
Get the result by converting the resulting response to array. The same below

Array(
    [status] => 204
    [header] => Array(
        [date] => Thu, 02 Jul 2015 02:29:12 GMT
        [content-length] => 0
        [connection] => close
        [server] => AliyunOSS
        [x-oss-request-id] => 5594A1F858DB3AB75215E0C4
    )

    [body] =>
)
```

Anti-leech Protection (Referer)

- Setting Referer anti-leech protection
    - Sample Code

```
$bucket_name = 'bucket name';
$is_allow_empty_referer = true;
$referer_list = array(
```

```
        'http://aliyun.com',
        'http://sina.com.cn'
    );
    $options = null;
    $response = $client->set_bucket_referer($bucket_name,$is_allow_empty_referer,$referer_list,$options);
```

- Parameter description

```
    $bucket_name: a required parameter. Must comply with the bucket naming rules
    $is_allow_empty_referer: a required parameter. It determines whether the referer can be blank, true by default
    $referer_list: a required parameter. It indicates the white list of allowed referers. Note that each record must
begin with http://
    $options: optional
```

- Response result

```
Get the result by converting the resulting response to array. The same below

    Array(
        [status] => 200
        [header] => Array(
            [date] => Thu, 02 Jul 2015 03:30:46 GMT
            [content-length] => 0
            [connection] => close
            [server] => AliyunOSS
            [x-oss-request-id] => 5594B06658DB3AB75215E9EC
        )

        [body] =>
    )
```

- Getting referer settings
  - Sample Code

```
    $bucket_name = 'bucket name';
    $options = null;
    $response = $client->get_bucket_referer($bucket_name,$options);
```

- Parameter description

```
    $bucket_name: a required parameter. Must comply with the bucket naming rules
    $options: optional
```

- Response result

```
Get the result by converting the resulting response to array. The same below
```

```
Array(
    [status] => 200
    [header] => Array(
        [date] => Thu, 02 Jul 2015 03:30:46 GMT
        [content-type] => application/xml
        [content-length] => 248
        [connection] => close
        [server] => AliyunOSS
        [x-oss-request-id] => 5594B06658DB3AB75215E9F2
    )

    [body] => Array(
        [RefererConfiguration] => Array(
            [AllowEmptyReferer] => true
            [RefererList] => Array(
                [Referer] => Array(
                    [0] => http://aliyun.com
                    [1] => http://sina.com.cn
                )
            )
        )
    )
)
```

URL Signature Operations

- Getting Get signed URLs
    - Sample Code

```
$bucket_name = 'bucket name';
$object_name = 'object name';
$timeout = 3600;
$options = null;
$signed_url = $client->get_sign_url($bucket_name,$object_name,$timeout,$options);
```

- Parameter description

```
$bucket_name: a required parameter. Must comply with the bucket naming rules
$object_name: a required parameter. Must comply with the object naming rules
$timeout: required parameter; the expiration time
$options: optional
```

- Response result

```
http://common-bucket.oss-cn-
shanghai.aliyuncs.com/my_get_file.log?OSSAccessKeyId=ACSb***&Expires=1435820652&
Signature=AW5z87zmaLulEmvMzf6ZOUrVboE%3D
```

- Getting Get or Put signed URLs

- Sample Code

```
$bucket_name = 'bucket name';
$object_name = 'object name';
$timeout = 3600;
$method = ALIOSS::OSS_HTTP_GET;
$options = null;
$signed_url = $client->get_sign_url($bucket_name,$object_name,$timeout,$method,$options);
```

- Parameter description

```
$bucket_name: a required parameter. Must comply with the bucket naming rules
$object_name: a required parameter. Must comply with the object naming rules
$timeout: required parameter; the expiration time
$method: a required parameter. It specifies the method type, currently GET or PUT
$options: optional
```

- Response result

```
http://common-bucket.oss-cn-
shanghai.aliyuncs.com/my_get_file.log?OSSAccessKeyId=ACSb***&Expires=1435820652&
Signature=AW5z87zmaLulEmvMzf6ZOUrVboE%3D
```

C-SDK

Preface

Introduction

This document introduces the installation and use of the OSS C SDK (for version 0.0.2 in particular).This document assumes that you have already subscribed to the AliCloud OSS service and created an Access Key ID and Access Key Secret.In the document, ID represents the Access Key ID and KEY indicates the Access Key Secret. If you have not yet subscribed to or do not know about the OSS service, please log into the OSS Product Homepage for more help.

Version Revisions

OSS C SDK (2015-08-17) Version 0.04

Updates:

1. Added support for keeplive persistent connections
2. Added support for lifecycle settings

OSS C SDK (2015-07-08) Version 0.03

Updates:

1. Added oss_append_object_from_buffer interface to append buffer content to the object
2. Added oss_append_object_from_file interface to append file content to the object

OSS C SDK (2015-06-10) Version 0.0.2

Updates:

1. Added the oss_upload_part_copy interface to support the Upload Part Copy method
2. Enabled access to OSS through temporary authorization in STS service

OSS C SDK (2015-05-28) Version 0.0.1

Updates:

1. Added the oss_create_bucket interface to create OSS buckets
2. Added the oss_delete_bucket interface to delete OSS buckets
3. Added the oss_get_bucket interface to get OSS buckets' ACLs
4. Added the oss_list_object interface to list objects in OSS buckets
5. Added oss_put_object_from_buffer interface to upload buffer content to the object
6. Added oss_put_object_from_file interface to upload file content to the object
7. Added oss_get_object_to_buffer interface to download object content to the buffer
8. Added oss_get_object_to_file interface to download object content to the file
9. Added the oss_head_object interface, to get objects' user meta information
10. Added the oss_delete_object interface to delete objects
11. Added the oss_copy_object interface to copy objects
12. Added the oss_init_multipart_upload interface to initialize multipart uploads
13. Added oss_upload_part_from_buffer interface to upload buffer content to the part
14. Added oss_upload_part_from_file interface to upload file content to the part
15. Added the oss_list_upload_part interface to retrieve information for all uploaded parts
16. Added the oss_complete_multipart_upload interface for multipart uploading
17. Added the oss_abort_multipart_upload interface to cancel multipart upload tasks
18. Added the oss_list_multipart_upload interface to get all multipart upload tasks in the bucket
19. Added the oss_gen_signed_url interface to generate a signed URL

20. Added the oss_put_object_from_buffer_by_url interface to upload the buffer content to the object using the URL signature method
21. Added the oss_put_object_from_file_by_url interface to upload the file content to the object using the URL signature method
22. Added the oss_get_object_to_buffer_by_ur interface download the object content to the buffer using the URL signature method
23. Added the oss_get_object_to_file_by_ur interface to download the object content to the file using the URL signature method
24. Added the oss_head_object_by_url interface to get objects' user meta information using the URL signature method

Installation

Steps:

1. Download the Object Storage Service C SDK from the official website
2. Decompress the file
3. After decompression, use the make command to generate the liboss_c_sdk.a static library from the files in the folder
4. Copy the liboss_c_sdk.a static library to the third-party library of your project
5. After completing the steps above, you can use the OSS C SDK in the project

The OSS C SDK uses the autoconf and automake compiling methods and cURL for network operations. Because you need to use HMAC to make digital signatures for authorization, this is dependent on the OpenSSL library. In addition, the OSS C SDK uses the apr library as its underlying data structure and uses libxml2 to parse xml format request responses. The OSS C SDK does not come with these external libraries. Therefore, prior to using the OSS C SDK, you must make sure the required external libraries are already installed in your development environment and that their header file and library file directories have been added in the project settings. This document does not discuss the installation of these third-party libraries, so please find the relevant materials on your own.

In an embedded environment, we suggest using -Os optimization options when installing third-party libraries. In addition, please refer to the following links to reduce the space occupied by third-party libraries. For optimization options during Curl installation, refer to http://www.cokco.cn/thread-11777-1-1.html; for optimization options during libxml2 installation, refer to http://curl.haxx.se/docs/install.html.

During project construction, if an environment-related compilation or link error occurs, please make sure these options are configured correctly and that the libraries they depend on are correctly installed.

Quick Start

In this chapter, you will learn how to use the basic functions of the OSS C SDK.

Step-1. Initializing the OSS C SDK Runtime Environment

When using the OSS C SDK, you must first initialize the runtime environment. When you wish to stop using it, you must clear the runtime environment. The following code demonstrates the initialization of the OSS C SDK runtime environment:

```
int main(int argc, char *argv[])
{
    //aos_http_io_initialize first
    if (aos_http_io_initialize("oss_test", 0) != AOSE_OK) {
        exit(1);
    }

    //use OSS C SDK api to access OSS
    ...

    aos_http_io_deinitialize();
    return 0;
}
```

aos_http_io_initialize initializes the OSS C SDK runtime environment. The first parameter can be used to customize the user agent content. aos_http_io_deinitialize clears the OSS C SDK runtime environment.

Step-2. Initializing an oss_request_options

The OSS operations of the OSS C SDK are completed using the oss_request_options_t structure. The following code creates an oss_request_options object:

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;

aos_pool_create(&p, NULL);
//init_ oss_request_options
oss_request_options = oss_request_options_create(p);
oss_request_options->config = oss_config_create(oss_request_options ->pool);
aos_str_set(&oss_request_options ->config->host,  oss_endpoint);
oss_request_options->config->port=oss_port;
aos_str_set(&oss_request_options ->config->id,  access_key_id);
aos_str_set(&oss_request_options ->config->key,  access_key_secret );
oss_request_options ->config->is_oss_domain = is_oss_domain;
```

```
oss_request_options ->ctl = aos_http_controller_create(oss_request_options ->pool, 0);
```

In the above code, the variables access_key_id and access_key_secret are allocated to the user by the system. They are called the ID pair and used to identify the user. They are used to perform signature verification when accessing OSS. For more information on oss_request_options, refer to oss_request_options.

Step-3. Creating Buckets

Buckets are the OSS global namespace. They are equivalent to a data container and can store numerous objects. You can create a bucket with the following code:

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *resp_headers;
oss_acl_e oss_acl = OSS_ACL_PRIVATE;
aos_string_t bucket;
char *bucket_name = "<your bucket name>";

aos_pool_create(&p, NULL);
// init_ oss_request_options
...
aos_str_set(&bucket, bucket_name);
resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);
s = oss_create_bucket(oss_request_options, &bucket, oss_acl, &resp_headers);
aos_pool_destroy(p);
```

For the Bucket naming rules, refer to the naming rules in Bucket.

Step-4. Uploading Objects

Objects are the basic data elements in OSS. You can simply think of them as files. The code below will upload an object:

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *headers;
aos_table_t *resp_headers;
aos_string_t bucket;
aos_string_t object;
char *bucket_name =  "<your bucket name>";
```

```
char *object_name =  "<your object name>";
char *data = "<your object content>";
aos_list_t buffer;
aos_buf_t *content;

aos_pool_create(&p, NULL);
// init_ oss_request_options
...
aos_str_set(&object, object_name);
aos_str_set(&bucket, bucket_name);
headers = aos_table_make(p, 1);
apr_table_set(headers, "x-oss-meta-author", "oss"); //object user meta
resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);

//read object content into buffer
aos_list_init(&buffer);
content = aos_buf_pack(oss_request_options->pool, data, strlen(data));
aos_list_add_tail(&content->node, &buffer);
s = oss_put_object_from_buffer (oss_request_options, &bucket, &object, &buffer, headers, &resp_headers);

aos_pool_destroy(p);
```

For object naming rules, refer to the naming rules in Object. For more information on uploading objects, refer to uploading objects in Object.

Step-5. Listing All Objects

When you complete a series of uploads, you may need to view which objects are in a bucket. This can be done with the following program:

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *resp_headers;
aos_string_t bucket;
char *bucket_name = "<your bucket name>";
oss_list_object_params_t *params;

aos_pool_create(&p, NULL);
// init_ oss_request_options
...
aos_str_set(&bucket, bucket_name);
resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);

params = oss_create_list_object_params(p);
params->max_ret = 10;
aos_str_set(&params->prefix, "object_prefix");
aos_str_set(&params-> delimiter, "object_ delimiter");
```

```
aos_str_set(&params-> marker, "object_marker");

s = oss_list_object(oss_request_options, &bucket, params, &resp_headers);;

aos_pool_destroy(p);
```

For more flexible parameter configurations, refer to [List Bucket Objects] in Object.

Step-6. Retrieving a specified object

You can refer to the code below to easily retrieve an object:

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *headers;
aos_table_t *resp_headers;
aos_string_t bucket;
aos_string_t object;
char *bucket_name =  "<your bucket name>";
char *object_name =  "<your object name>";
aos_list_t buffer;

aos_pool_create(&p, NULL);
// init_ oss_request_options
...
aos_str_set(&object, object_name);

aos_str_set(&bucket, bucket_name);
headers = aos_table_make(p, 0);
resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);

//get object content into buffer
aos_list_init(&buffer);
s = oss_get_object_to_buffer(oss_request_options, &bucket, &object, headers, &buffer, &resp_headers);

aos_pool_destroy(p);
```

You can read the user meta information of the object from the response header.

oss_request_options

When using the OSS C SDK to perform OSS operations, you must initialize oss_request_options. Here, the config variable is used to store basic OSS access information, e.g., the OSS domain name, OSS

port No., and the user’ s accessKeyId/accessKeySecret. The is_oss_domain variable specifies whether OSS access uses a third-level domain name (for CNAME, OSS access uses a second-level domain name). The ctl variable initializes the OSS C SDK’ s OSS access control information. By setting this variable, you can perform flow control on OSS access.

Initializing oss_request_options

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;

aos_pool_create(&p, NULL);
//init_ oss_request_options
oss_request_options = oss_request_options_create(p);
oss_request_options ->config = oss_config_create(oss_request_options ->pool);
aos_str_set(&oss_request_options ->config->host, oss_endpoint);
oss_request_options ->config->port=oss_port;
aos_str_set(&oss_request_options ->config->id, access_key_id);
aos_str_set(&oss_request_options ->config->key, access_key_secret );
oss_request_options ->config->is_oss_domain = is_oss_domain;
oss_request_options ->ctl = aos_http_controller_create(oss_request_options ->pool, 0);
```

Here, the value corresponding to the region of the bucket to be accessed must be entered for oss_endpoint.

Configuring oss_request_options

If you wish to set the OSS C SDK’ s underlying libcurl communication parameters, you can set the ctl member variable in oss_request_options.

Bucket

OSS uses buckets as the namespaces of user files and also as the management objects for advanced functions such as charging, permission control, and log recording. The bucket name must be globally unique in the entire OSS and cannot be changed. Every object stored on the OSS must be included in a bucket. One application, such as an image sharing website, can correspond to one or more buckets. A user can create a maximum of 10 buckets, but there is no limit on the number and total size of objects in each bucket, so the user does not have to consider data scalability.

Naming Rules

The bucket naming rules are as follows:

- It can only contain lower-case letters, numbers, and dashes (-).
- It must start with a lower-case letter or number.
- The length must be 3-63 bytes

Creating Buckets

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *resp_headers;
oss_acl_e oss_acl = OSS_ACL_PRIVATE;
aos_string_t bucket;
char *bucket_name = "<your bucket name>";

aos_pool_create(&p, NULL);
// init_ oss_request_options
...
aos_str_set(&bucket, bucket_name);
resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);
s = oss_create_bucket(oss_request_options, &bucket, oss_acl, &resp_headers);

aos_pool_destroy(p);
```

To create a bucket, you must enter the bucket name and ACL. Because bucket names are globally unique, do your best to ensure your bucket names are not the same as other people's. ACL currently supports the values private, public-read, and public-read-write. For information on permissions, please refer to OSS Access Control. If a bucket of the same name already exists and belongs to the owner, this operation can overwrite the bucket's ACL settings.

Retrieving Bucket ACL

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *resp_headers;
aos_string_t bucket;
char *bucket_name = "<your bucket name>";
aos_string_t oss_acl;

aos_pool_create(&p, NULL);
// init_ oss_request_options
...
aos_str_set(&bucket, bucket_name);
```

```
resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);
s = oss_get_bucket_acl(oss_request_options, &bucket,& oss_acl, &resp_headers);

aos_pool_destroy(p);
```

Deleting Buckets

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *resp_headers;
aos_string_t bucket;
char *bucket_name = "<your bucket name>";

aos_pool_create(&p, NULL);
// init_ oss_request_options
...
aos_str_set(&bucket, bucket_name);
resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);
s = oss_delete_bucket (oss_request_options, &bucket, &resp_headers);

aos_pool_destroy(p);
```

Please note that if the bucket is not empty (i.e., bucket contains objects or multipart upload fragments), it cannot be deleted. You must delete all objects and fragments in a bucket before deleting the bucket.

Object

In OSS, objects are the basic data units for user operation.The maximum size of a single object may vary depending on the data uploading mode. The size of an object cannot exceed 5 GB in the Put Object mode or 48.8 TB in the multipart upload mode. An object includes the key, meta, and data. The key is the object name; meta is the user's description of the object, composed of a series of name-value pairs; and data is the object data.

Naming Rules

Object naming rules:

- It uses UTF-8 encoding
- The length must be 1-1023 bytes

- It cannot start with "/" or "\"
- It cannot contain "\r" or "\n" line breaks

Uploading Objects

Simple Upload

Uploads data from the memory to OSS

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *headers;
aos_table_t *resp_headers;
aos_string_t bucket;
aos_string_t object;
char *bucket_name =  "<your bucket name>";
char *object_name =  "<your object name>";
char *data = "<your object content>";
aos_list_t buffer;
aos_buf_t *content;

aos_pool_create(&p, NULL);
// init_ oss_request_options
...
aos_str_set(&object, object_name);

aos_str_set(&bucket, bucket_name);
headers = aos_table_make(p, 0);
resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);

//read object content into buffer
aos_list_init(&buffer);
content = aos_buf_pack(oss_request_options->pool, data, strlen(data));
aos_list_add_tail(&content->node, &buffer);
s = oss_put_object_from_buffer (oss_request_options, &bucket, &object, &buffer, headers, &resp_headers);
aos_pool_destroy(p);
```

Uploads local files to OSS

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *headers;
aos_table_t *resp_headers;
aos_string_t bucket;
```

```
aos_string_t object;
char *bucket_name =  "<your bucket name>";
char *object_name =  "<your object name>";
char *data = "<your object content>";
char *filename ="<your local filename>";
aos_string_t local_file;

aos_pool_create(&p, NULL);
// init_ oss_request_options
...
aos_str_set(&object, object_name);
aos_str_set(&local_file,filename);
aos_str_set(&bucket, bucket_name);
headers = aos_table_make(p, 0);
resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);

s = oss_put_object_from_file (oss_request_options, &bucket, &object, &local_file, headers, &resp_headers);

aos_pool_destroy(p);
```

When uploading in this manner, the largest file cannot exceed 5G. If the size exceeds this, use multipartupload to upload.

Creating Simulated Folders

The OSS service does not use folders. All elements are stored as objects. However, users can create simulated folders using the following code:

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *headers;
aos_table_t *resp_headers;
aos_string_t bucket;
aos_string_t object;
char *bucket_name =  "<your bucket name>";
char *object_name =  "folder_name/";
char *data = "";
aos_list_t buffer;
aos_buf_t *content;

aos_pool_create(&p, NULL);
// init_ oss_request_options
...

aos_str_set(&bucket, bucket_name);
aos_str_set(&object, object_name);
headers = aos_table_make(p, 0);
resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);
```

```
//read object content into buffer
...

s = oss_put_object_from_buffer (oss_request_options, &bucket, &object, &buffer, headers, &resp_headers);

aos_pool_destroy(p);
```

Creating a simulated folder is in fact creating an object with a size of 0. This object can also be uploaded and downloaded. The console will display any object ending with "/" as a folder. Therefore, users can create simulated folders this way. For accessing folders, refer to the folder simulation function

Setting the Object's Http Header

The OSS service allows users to customize the object Http Header. The following code sets the expiration time for the object:

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *headers;
aos_table_t *resp_headers;
aos_string_t bucket;
aos_string_t object;
char *bucket_name =  "<your bucket name>";
char *object_name =  "<your object name>"
char *data = "<your object content>";
aos_list_t buffer;
aos_buf_t *content;

aos_pool_create(&p, NULL);
// init_ oss_request_options
...

aos_str_set(&bucket, bucket_name);
aos_str_set(&object, object_name);

//set http header
headers = aos_table_make(p, 1);
apr_table_set(headers, " Expires", " Fri, 28 Feb 2012 05:38:42 GMT");

resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);

//read object content into buffer
...
aos_pool_destroy(p);
```

You can set the HTTP header to:Cache-Control, Content-Disposition, Content-Encoding, and

Expires.For details on the headers, please see RFC2616.

Setting User Meta

The OSS allows users to define meta information to describe the object. For example:

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *headers;
aos_table_t *resp_headers;
aos_string_t bucket;
aos_string_t object;
char *bucket_name =  "<your bucket name>";
char *object_name =  "<your object name>"
char *data = "<your object content>";
aos_list_t buffer;
aos_buf_t *content;

aos_pool_create(&p, NULL);
// init_ oss_request_options
...

aos_str_set(&bucket, bucket_name);
aos_str_set(&object, object_name);
headers = aos_table_make(p, 1);
//set user meta
apr_table_set(headers, "x-oss-meta-author", "oss");
resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);
//read object content into buffer
...

aos_pool_destroy(p);
```

A single object can have multiple similar parameters, but the total size of all user meta cannot exceed 2 KB.

> NOTE:The user meta name is not case sensitive. For instance, when a user uploads an object and defines the metadata name as "Name", the parameter stored in the header will be: "x-oss-meta-name". Therefore, when accessing the object, just use parameters named "name". However, if the stored parameter is "name", and no information can be found for the parameter, the system will return "Null"

Append Object

Append Object is used to upload files in appending mode.The type of the objects created with the

Append Object operation is Appendable Object, and the type of the objects uploaded with the Put Object operation is Normal Object.

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *headers;
aos_table_t *resp_headers;
aos_string_t bucket;
aos_string_t object;
char *bucket_name =  "<your bucket name>";
char *object_name =  "<your object name>";
char *data = "<your object content>";
char *data1 = "<your object content>";
aos_list_t buffer;
aos_buf_t *content;
aos_buf_t *content1;
int64_t position = 0;
aos_pool_create(&p, NULL);
// init_ oss_request_options
...

aos_str_set(&object, object_name);
aos_str_set(&bucket, bucket_name);
headers = aos_table_make(p, 0);
resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);

//append object from start position
aos_list_init(&buffer);
content = aos_buf_pack(oss_request_options ->pool, data, strlen(data));
aos_list_add_tail(&content->node, &buffer);
s = oss_append_object_from_buffer (oss_request_options, &bucket, &object, position, &buffer, headers,
&resp_headers);

//append object from not start position
headers = aos_table_make(p, 0);
position = strlen(data);
aos_list_init(&buffer);
content = aos_buf_pack(oss_request_options->pool, data1, strlen(data1));
aos_list_add_tail(&content1->node, &buffer);
s = oss_append_object_from_buffer (oss_request_options, &bucket, &object, position, &buffer, headers,
&resp_headers);
aos_pool_destroy(p);



os_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *headers;
aos_table_t *resp_headers;
```

```
aos_string_t bucket;
aos_string_t object;
char *bucket_name =  "<your bucket name>";
char *object_name =  "<your object name>";
char *data = "<your object content>";
char *filename ="<your local filename>";
aos_string_t append_file;
int64_t position = 0;

aos_pool_create(&p, NULL);
// init_ oss_request_options
...
aos_str_set(&object, object_name);
aos_str_set(&append_file,filename);
aos_str_set(&bucket, bucket_name);
headers = aos_table_make(p, 0);
resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);

s = oss_append_object_from_file (oss_request_options, &bucket, &object, position, &append_file, headers,
&resp_headers);

aos_pool_destroy(p);
```

1. Append Object is not applicable to a non-appendable object.For example, if a normal object with the same name already exists and the Append Object operation is still performed, the system returns the 409 message and the error code ObjectNotAppendable.
2. If you perform the Put Object operation on an existing appendable object, this appendable object is overwritten by the new object, and the type of this object is changed to Normal Object.
3. After the Head Object operation is performed, the system returns x-oss-object-type, which indicates the type of the object. If the object is an appendable object, the value of x-oss-object-type is Appendable. For an appendable object, after the Head Object operation is performed, the system also returns x-oss-next-append-position and x-oss-hash-crc64ecma.
4. You can neither use Copy Object to copy an appendable object, nor change the server-side encryption attribute of this object. You can, however, use Copy Object to change the customized metadata.
5. In List Objects requests' XML responses, this will set the Appendable Object's Type to Appendable.

Multipart Upload

OSS allows users to split an object into several requests for uploading to the server. Concerning multipart upload, refer to the Object Multipart Upload section in MultipartUpload.

List Bucket Objects

Object information is saved in the params object_list. You can use aos_list_for_each_entry to view

detailed information for each object. NOTE:By default, if a bucket contains more than 100 objects, the first 100 will be returned and the IsTruncated parameter in the returned results will be true. The returned NextMarker can be used as the start point for next data access.The number of object entries returned can be increased by modifying the MaxKeys parameter or using the Marker parameter for separate access.

Extended Parameters

Generally, the ListObjectsRequest parameter provides more powerful functions. For example:

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *resp_headers;
aos_string_t bucket;
char *bucket_name = "<your bucket name>";
oss_list_object_params_t *params;

aos_pool_create(&p, NULL);
// init_ oss_request_options
...
aos_str_set(&bucket, bucket_name);
resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);

params = oss_create_list_object_params(p);
params->max_ret = 10;
aos_str_set(&params->prefix, "pic");
aos_str_set(&params-> delimiter, "/");
aos_str_set(&params-> marker, "");

s = oss_list_object(oss_request_options, &bucket, params, &resp_headers);;

aos_pool_destroy(p);
```

The above code lists bucket CommonPrefixes with the prefix "pic" and ending in "/". For example, "pic-people/". Settable parameter names and their functions:

Name	Function
Delimiter	Used to group object name characters. All objects whose names contain the specified prefix and that appear between the Delimiter characters for the first time are used as a group of elements: CommonPrefixes.
Marker	Sets up the returned results to begin from the first entry after the Marker in alphabetical order.
MaxKeys	Limits the maximum number of objects returned for one request. If not specified, the

	default value is 100. The MaxKeys value cannot exceed 1000.
Prefix	requires the returned object key to be prefixed with prefix. Note that the keys returned from queries using a prefix will still contain the prefix.

Retrieving Objects

Reading Objects

The input stream can be used to get and store the object content into an object or the memory. Object header information can be obtained through resp_headers. It contains the ETag, Http Header, and custom metadata defined at the time the object was uploaded.

Parameter	Description
Range	Specifies the range of file transfer.
ModifiedSinceConstraint	If the specified time is earlier than the actual modification time, the file is transmitted normally. Otherwise, the system throws the 304 Not Modified exception.
UnmodifiedSinceConstraint	If the specified time is the same as or later than the actual modification time, the file is transmitted normally.Otherwise, the system throws the 412 precondition failed exception
MatchingETagConstraints	Imports an ETag group. If the imported expected ETag matches the object's ETag, the file is transmitted normally.Otherwise, the system throws the 412 precondition failed exception
NonmatchingEtagConstraints	Imports an ETag group. If the imported ETag does not match the object's ETag, the file is transmitted normally.Otherwise, the system throws the 304 Not Modified exception.
ResponseHeaderOverrides	Customizes some headers in the OSS return request.

We can set Range to return the object range. We can use this function for segmented file multipart download and resumable data transfer.

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *headers;
```

```
aos_table_t *resp_headers;
aos_string_t bucket;
aos_string_t object;
char *bucket_name =  "<your bucket name>";
char *object_name =  "<your object name>";
aos_list_t buffer;

aos_pool_create(&p, NULL);
// init_ oss_request_options
...

aos_str_set(&bucket, bucket_name);
aos_str_set(&object, object_name);
headers = aos_table_make(p, 1);
//Sets the range and reads the specified range from the file
   apr_table_set(headers, "Range", " bytes=20-100");
resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);

//get object content into buffer
aos_list_init(&buffer);
s = oss_get_object_to_buffer(oss_request_options, &bucket, &object, headers, &buffer, &resp_headers);

aos_pool_destroy(p);
```

Directly Downloading Objects to Files We can use the code below to directly download objects to a specified file:

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *headers;
aos_table_t *resp_headers;
aos_string_t bucket;
aos_string_t object;
char *bucket_name =  "<your bucket name>";
char *object_name =  "<your object name>";
int object_name_len;
char *download_filename
aos_string_t download_file;

aos_pool_create(&p, NULL);
// init_ oss_request_options
...

aos_str_set(&bucket, bucket_name);
aos_str_set(&object, object_name);
aos_str_set(&download_file, download_filename);
headers = aos_table_make(p, 0);
resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);
s = oss_get_object_to_file (oss_request_options, &bucket, &object, headers, &download_file,  &resp_headers);
```

```
aos_pool_destroy(p);
```

Deleting Objects

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *resp_headers;
aos_string_t bucket;
aos_string_t object;
char *bucket_name =  "<your bucket name>";
char * object_name =  "<your object name>";

aos_pool_create(&p, NULL);
// init_ oss_request_options
...

aos_str_set(&bucket, bucket_name);
aos_str_set(&object, object_name);

resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);

s = oss_delete_object (oss_request_options, &bucket, &object, ,  &resp_headers);

aos_pool_destroy(p);
```

Copying Objects

You can copy an object with operation permissions within the same region. We would like to remind users that, when copying an object larger than 1G, we suggest using the Upload Part Copy method.

Copying One Object

Using the copyObject method, we can copy a single object. The code is as follows:

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t * headers;
aos_table_t *resp_headers;
aos_string_t source_bucket;
aos_string_t source_object;
aos_string_t dest_bucket;
aos_string_t dest_object;
char *source_bucket_name =  "<your bucket name>";
char *source_object_name =  "<your object name>";
```

```
char *dest_bucket_name =  "<your bucket name>";
char *dest_object_name =  "<your object name>";

aos_pool_create(&p, NULL);
// init_ oss_request_options
...

aos_str_set(&source_bucket, bucket_name);
aos_str_set(&source_object, source_object_name);
aos_str_set(&dest_bucket, dest_bucket_name);
aos_str_set(&dest_object, dest_object_name);

headers = aos_table_make(p, 0);
resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);

s = oss_copy_object (oss_request_options, &source_bucket, &source_object, &dest_bucket, &dest_object ,headers,
&resp_headers);

aos_pool_destroy(p);
```

Note that the source and destination buckets must be in the same region.

Multipart Upload

Besides using the putObject interface to upload files to OSS, the OSS also provides a Multipart Upload mode.You can apply the Multipart Upload mode in the following scenarios (but not limited to the following):

- Where breakpoint uploads are needed.
- Uploading an object larger than 100MB.
- In poor network conditions, when the connection with the OSS server is frequently broken.
- Stream uploading an object. When, before uploading the file, you cannot determine its size.

Step-By-Step Multipart Upload

Initialization

Initializes a single multipart upload task

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t * headers;
```

```
aos_table_t *resp_headers;
aos_string_t bucket;
aos_string_t object;
char *bucket_name =  "<your bucket name>";
char * object_name =  "<your object name>";
aos_string_t upload_id;

aos_pool_create(&p, NULL);
// init_ oss_request_options
...
aos_str_set(&bucket, bucket_name);
aos_str_set(&object,object_name);

headers = aos_table_make(p, 0);
resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);
s = oss_init_multipart_upload (oss_request_options, &bucket, &object, headers, &upload_id,

aos_pool_destroy(p);
```

The returned results contain the upload_id. This is the unique identifier of a multipart upload task. We will use this in subsequent operations.

Upload Part Local Upload

Next, we will multipart upload the local file. Let us assume that there is one file in the local path /path/to/file.zip. Because it is large, we want to multipart upload it to OSS.

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *resp_headers;
aos_string_t bucket;
aos_string_t object;
char *bucket_name =  "<your bucket name>";
char * object_name =  "<your object name>";
aos_string_t upload_id;
char *filename = "<local filename>";
oss_upload_file_t *upload_file;

aos_pool_create(&p, NULL);
// init_ oss_request_options
...

aos_str_set(&bucket, bucket_name);
aos_str_set(&object, object_name);

upload_file = oss_create_upload_file(p);
aos_str_set(&upload_file->filename, filename);
upload_file->file_pos = 0;
upload_file->file_last = 200 * 1024; //200KB
```

```
resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);
s = oss_upload_part_from_file (oss_request_options, &bucket, &object, &upload_id, part_num,  upload_file,
&resp_headers);

aos_pool_destroy(p);
```

The main idea of this program is to call the oss_upload_part_from_file method to upload each part. However, you must note the following:

- The oss_upload_part_from_file method requires that all parts except the last one must be larger than 100KB.
- In order to ensure that the data transmitted over the network is free from errors, we strong recommend that the user include meta: content-md5 in the request when uploading parts. After the OSS receives data, it uses this MD5 value to verify the correctness of the uploaded data. If it is not consistent, OSS returns the InvalidDigest error code.
- The part number range is 1~10000. If the part number exceeds this range, the OSS will return the InvalidArgument error code.
- When each part is uploaded, it will take the stream to the location corresponding to the start of the next part.

Completing Multipart Uploads

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *resp_headers;
aos_string_t bucket;
aos_string_t object;
char *bucket_name =  "<your bucket name>";
char * object_name =  "<your object name>";
aos_string_t upload_id;
aos_list_t complete_part_list;

aos_pool_create(&p, NULL);
// init_ oss_request_options
...

aos_str_set(&bucket, bucket_name);
aos_str_set(&object, object_name);

//build complete_part_list
...

resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);
s = oss_complete_multipart_upload (oss_request_options, &bucket, &object, &upload_id, &complete_part_list,
&resp_headers);
```

```
aos_pool_destroy(p);
```

Canceling Multipart Upload Tasks

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *resp_headers;
aos_string_t bucket;
aos_string_t object;
char *bucket_name =  "<your bucket name>";
char * object_name =  "<your object name>";
aos_string_t upload_id;

aos_pool_create(&p, NULL);
// init_ oss_request_options
...

aos_str_set(&bucket, bucket_name);
aos_str_set(&object, object_name);

resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);
s = oss_abort_multipart_upload (oss_request_options, &bucket, &object, &upload_id, &resp_headers);

aos_pool_destroy(p);
```

When a Multipart Upload event is aborted, you cannot use this Upload ID to perform any operations and the uploaded parts of data will be deleted. Get All Multipart Upload Tasks in the Bucket

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *resp_headers;
aos_string_t bucket;
aos_string_t object;
char *bucket_name =  "<your bucket name>";
char * object_name =  "<your object name>";
aos_string_t upload_id;

aos_pool_create(&p, NULL);
// init_ oss_request_options
...

aos_str_set(&bucket, bucket_name);
aos_str_set(&object, object_name);

resp_headers = aos_table_make(p, 0);
```

```
s = aos_status_create(p);
s = oss_abort_multipart_upload (oss_request_options, &bucket, &object, &upload_id, &resp_headers);

aos_pool_destroy(p);
```

All upload event information is stored in the oss_list_multipart_upload_params_t upload_list, which can be traversed using aos_list_for_each_entry.

Getting Information for All Uploaded Parts

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *resp_headers;
aos_string_t bucket;
aos_string_t object;
char *bucket_name = "<your bucket name>";
char *object_name = "<your object name>";
char *utf8_object_name;
aos_string_t upload_id;
oss_list_upload_part_params_t *params;

aos_pool_create(&p, NULL);
// init_ oss_request_options
...
//utf8 encode object name
...

aos_str_set(&bucket, bucket_name);
aos_str_set(&object, utf8_object_name);

resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);

params = params = oss_create_list_upload_part_params(p);
params->max_ret = 10;

s = oss_list_upload_part (oss_request_options, &bucket, &object, &upload_id,params, &resp_headers);;

aos_pool_destroy(p);
```

All part information is stored in the oss_list_upload_part_params_t part_list, which can be traversed using aos_list_for_each_entry.

Multipart Upload Copy

Using Upload Part Copy, we copy data from an existing object to upload an object.When copying an object larger than 500MB, we suggest using the Upload Part Copy method.

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_string_t upload_id;
oss_list_upload_part_params_t *list_upload_part_params;
oss_upload_part_copy_params_t *upload_part_copy_params1;
aos_table_t *headers;
aos_table_t *resp_headers;
aos_table_t *list_part_resp_headers;
aos_list_t complete_part_list;
oss_list_part_content_t *part_content;
oss_complete_part_content_t *complete_content;
aos_table_t *complete_resp_headers;
aos_status_t *s;
int part1 = 1;
char *source_bucket_name = "<your bucket name>";
char *dest_bucket_name = "<your bucket name>";
char *source_object_name = "<your source object name>";
char *dest_object_name = "<your dest object name>";
aos_string_t dest_bucket;
aos_string_t dest_object;
int64_t range_start1 = 0;
int64_t range_end1 = 6000000;//not less than 5MB

aos_pool_create(&p, NULL);
// init_ oss_request_options
...

//init mulitipart upload
//upload part copy part 1
upload_part_copy_params1 = oss_create_upload_part_copy_params(p);
aos_str_set(&upload_part_copy_params1->source_bucket, source_bucket_name);
aos_str_set(&upload_part_copy_params1->source_object, source_object_name);
aos_str_set(&upload_part_copy_params1->dest_bucket, dest_bucket_name);
aos_str_set(&upload_part_copy_params1->dest_object, dest_object_name);
aos_str_set(&upload_part_copy_params1->upload_id, upload_id.data);
upload_part_copy_params1->part_num = part1;
upload_part_copy_params1->range_start = range_start1;
upload_part_copy_params1->range_end = range_end1;
headers = aos_table_make(p, 0);
resp_headers = aos_table_make(p, 5);
s = oss_upload_part_copy(oss_request_options, upload_part_copy_params1, headers, &resp_headers);

// continue upload part copy like part 1
...

//list part
list_part_resp_headers = aos_table_make(p, 5);
list_upload_part_params = oss_create_list_upload_part_params(p);
list_upload_part_params->max_ret = 1000;
aos_list_init(&complete_part_list);

aos_str_set(&dest_bucket, dest_bucket_name);
aos_str_set(&dest_object, dest_object_name);
s = oss_list_upload_part(oss_request_options, &dest_bucket, &dest_object, &upload_id,
```

```
list_upload_part_params, &list_part_resp_headers);

aos_list_for_each_entry(part_content, &list_upload_part_params->part_list, node) {
  complete_content = oss_create_complete_part_content(p);
  aos_str_set(&complete_content->part_number, part_content->part_number.data);
  aos_str_set(&complete_content->etag, part_content->etag.data);
  aos_list_add_tail(&complete_content->node, &complete_part_list);
}

//complete multipart upload
complete_resp_headers = aos_table_make(p, 5);
s = oss_complete_multipart_upload(oss_request_options, &dest_bucket, &dest_object, &upload_id,
&complete_part_list, &complete_resp_headers);

aos_pool_destroy(p);
```

Lifecycle Management

OSS provides the object lifecycle management capability to manage objects for users. The user can configure the lifecycle of a bucket to define various rules for the bucket's objects. Currently, users can use rules to delete matching objects. Each rule is composed of the following parts:

- The object name prefix; this rule will only apply to objects with the matched prefix.
- Operation; the operation the user wishes to perform on the matched objects.
- Date or number of days; the user will execute the operation on the objects on the specified date or a specified number of days after the object's last modification time.

Setting Lifecycles

The lifecycle configuration rules are expressed by an xml segment.

```
<LifecycleConfiguration>
  <Rule>
    <ID>delete obsoleted files</ID>
    <Prefix>obsoleted/</Prefix>
    <Status>Enabled</Status>
    <Expiration>
      <Days>3</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

A single lifecycle Config can contain up to 1000 rules. Explanations of each field:

- The ID field is used to uniquely identify a rule (inclusion relations, such as abc and abcd, cannot exist between IDs).

- Prefix indicates the rules used for objects in the bucket with the specified prefix.
- Status indicates the status of this rule. The statuses are Enabled and Disabled, indicating if the rule is enabled or disabled.
- In the Expiration node, Days indicates that an object will be deleted a specified number of days after its last modification. Date indicates that objects will be deleted after the specified absolute time (the absolute time follows the ISO8601 format).

Using the following code, we can set the above lifecycle rules.

```
aos_pool_t *p;
int is_oss_domain = 1;
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *resp_headers;
aos_string_t bucket;
char *bucket_name = "<your bucket name>";
aos_list_t lifecycle_rule_list;
oss_lifecycle_rule_content_t *rule_content;

aos_pool_create(&p, NULL);
// init_ oss_request_options
...

aos_str_set(&bucket, bucket_name);

resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);
aos_list_init(&lifecycle_rule_list);
rule_content = oss_create_lifecycle_rule_content(p);
aos_str_set(&rule_content->id, " delete obsoleted files ");
aos_str_set(&rule_content->prefix, " obsoleted ");
aos_str_set(&rule_content->status, "Enabled");
rule_content->days = 3;
aos_list_add_tail(&rule_content->node, &lifecycle_rule_list);
s = oss_put_bucket_lifecycle(options, &bucket, &lifecycle_rule_list, &resp_headers);

aos_pool_destroy(p);
```

We can use the following code to retrieve the above lifecycle rules.

```
aos_pool_t *p;
int is_oss_domain = 1;
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *resp_headers;
aos_string_t bucket;
char *bucket_name = "<your bucket name>";
aos_list_t lifecycle_rule_list;
oss_lifecycle_rule_content_t *rule_content;
char *rule_id;
char *prefix;
char *status;
```

```
int days = INT_MAX;
char* date = "";

aos_pool_create(&p, NULL);
// init_ oss_request_options
...

aos_str_set(&bucket, bucket_name);

resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);
aos_list_init(&lifecycle_rule_list);
s = oss_get_bucket_lifecycle(options, &bucket, &lifecycle_rule_list, &resp_headers);
aos_list_for_each_entry(rule_content, &lifecycle_rule_list, node) {
    rule_id = apr_psprintf(p, "%.*s", rule_content->id.len, rule_content->id.data);
    prefix = apr_psprintf(p, "%.*s", rule_content->prefix.len, rule_content->prefix.data);
status = apr_psprintf(p,"%.*s", rule_content->status.len, rule_content->status.data);
date = apr_psprintf(p, "%.*s", rule_content->date.len, rule_content->date.data);
    days = rule_content->days;
}

aos_pool_destroy(p);
```

Using the following code, we can clear the lifecycle rules in a bucket.

```
aos_pool_t *p;
int is_oss_domain = 1;
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *resp_headers;
aos_string_t bucket;
char *bucket_name =  “<your bucket name>”;

aos_pool_create(&p, NULL);
// init_ oss_request_options
...

aos_str_set(&bucket, bucket_name);

resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);
s = oss_delete_bucket_lifecycle(options, &bucket, &resp_headers);

aos_pool_destroy(p);
```

Authorized Access

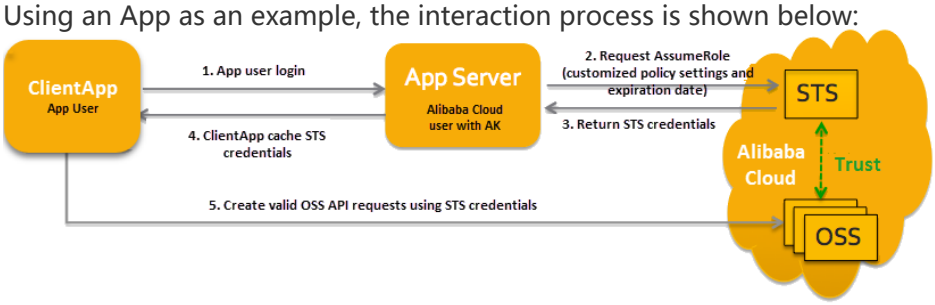
Using STS Service Temporary Authorization

Introduction

Through the AliCloud STS service, OSS can temporarily grant authorized access.AliCloud STS is a web service that provides a temporary access token to a cloud computing user. Using STS, you can grant access credentials to a third-party application or federated user (you can manage the user IDs) with customized permissions and validity periods. Third-party applications or federated users can use these access credentials to directly call the AliCloud product APIs or use the SDKs provided by AliCloud products to access the cloud product APIs.

- You do not need to expose you long-term key (AccessKey) to a third-party application and only need to generate an access token and send the access token to the third-party application. You can customize the access permission and validity of this token.
- You do not need to care about permission revocation issues. The access credential automatically becomes invalid when it expires.

Using an App as an example, the interaction process is shown below:



The solution is described in detail as follows:

1. Log in as the app user. App user IDs are managed by the client. Clients can customize the ID management system, and may also use external Web accounts or OpenID. For each valid app user, the AppServer can precisely define the minimum access permission.
2. The AppServer requests a security token (Security Token) from the STS. Before calling STS, the AppServer needs to determine the minimum access permission (described in policy syntax) of app users and the expiration time of the authorization. Then, the security token is obtained by calling the STS' AssumeRole interface.
3. The STS returns a valid access credential to the AppServer, where the access credential includes a security token, a temporary access key (AccessKeyId and AccessKeySecret), and the expiry time.
4. The AppServer returns the access credential to the ClientApp. The ClientApp caches this credential. When the credential becomes invalid, the ClientApp needs to request a new valid access credential from the AppServer. For example, if the access credential is valid for one hour, the ClientApp can request the AppServer to update the access credential every 30 minutes.
5. The ClientApp uses the access credential cached locally to request for AliCloud Service APIs. The ECS perceives the STS access credential, relies on STS to verify the credential, and correctly responds to the user request.

The key is to obtain a valid access credential by simply calling the STS interface AssumeRole.The

method can also be called by using the STS DSK. Clicking to View Details

Using STS Credentials to Construct Signed Requests

After obtaining the STS temporary credential, the user’ s client generates an oss_request_options using the contained Security Token and temporary access key (AccessKeyId, AccessKeySecret). Using an object upload as an example:

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_status_t *s;
aos_table_t *headers;
aos_table_t *resp_headers;
aos_string_t bucket;
aos_string_t object;
char *bucket_name = "<your bucket name>";
char *object_name = "<your object name>"
char *data = "<your object content>";
aos_list_t buffer;
aos_buf_t *content;

aos_pool_create(&p, NULL);

// init_oss_request_options using sts_token
oss_request_options = oss_request_options_create(p);
oss_request_options ->config = oss_config_create(oss_request_options ->pool);
aos_str_set(&oss_request_options ->config->host, oss_endpoint);
oss_request_options ->config->port=oss_port;
aos_str_set(&oss_request_options ->config->id, tmp_access_key_id);
aos_str_set(&oss_request_options ->config->key, tmp_access_key_secret );
aos_str_set(&oss_request_options ->config->sts_token, sts_token );
oss_request_options ->config->is_oss_domain = is_oss_domain;
oss_request_options ->ctl = aos_http_controller_create(oss_request_options ->pool, 0);

aos_str_set(&bucket, bucket_name);
aos_str_set(&object, object_name);
headers = aos_table_make(p, 0);
resp_headers = aos_table_make(p, 0);
s = aos_status_create(p);

aos_list_init(&buffer);
content = aos_buf_pack(oss_request_options ->pool, data, strlen(data));
aos_list_add_tail(&content->node, &buffer);
s = oss_put_object_from_buffer (oss_request_options, &bucket, &object, &buffer, headers, &resp_headers);
aos_pool_destroy(p);
```

URL Signature Authorization

You can provide users with a temporary access URL by generating a signed URL. During URL

generation, you can specify the URL expiration time to limit the duration of the user’ s access.

Generating a Signed URL

Generates a get object request URL signature.

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_http_request_t *req;
char *url_signed_str
char *bucket_name = "<your bucket name>";
char * object_name = "<your object name>";

aos_pool_create(&p, NULL);
// init_ oss_request_options
...

req = aos_http_request_create(p);
req->method = HTTP_GET;
url_str = gen_test_signed_url(oss_request_options,bucket_name, object_name, expire_time, req);

aos_pool_destroy(p);
```

Generates a put object request URL signature:

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_http_request_t *req;
char *url_signed_str;
char *bucket_name = "<your bucket name>";
char * object_name = "<your object name>";

aos_pool_create(&p, NULL);
// init_ oss_request_options
...

req = aos_http_request_create(p);
req->method = HTTP_PUT;
url_str = gen_test_signed_url(oss_request_options,bucket_name, object_name, expire_time, req);

aos_pool_destroy(p);
```

Using Signed URLs to Send Requests

Uses a signed URL to getobject

```
```

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_http_request_t *req;
aos_table_t *headers;
aos_table_t *resp_headers;
char *url_signed_str;
char *bucket_name = "<your bucket name>";
char * object_name = "<your object name>";
aos_string_t url;
aos_string_t download_file;
char *filename = "<local filename>"

aos_pool_create(&p, NULL);
// init_ oss_request_options
...

headers = aos_table_make(p, 0);
resp_headers = aos_table_make(p, 5);
req = aos_http_request_create(p);
req->method = HTTP_GET;
url_str = gen_test_signed_url(oss_request_options,bucket_name, object_name, expire_time, req);
aos_str_set(&url, url_str);
aos_str_set(&download_file, filename);
s = oss_get_object_to_file_by_url(oss_request_options, &url, headers,&download_file, &resp_headers);

aos_pool_destroy(p);
```

Uses a signed URL to putobject

```
aos_pool_t *p;
int is_oss_domain = 1;//Whether or not it uses a third-level domain name; can be initialized through the
is_oss_domain function
oss_request_options_t * oss_request_options;
aos_http_request_t *req;
aos_table_t *headers;
aos_table_t *resp_headers;
char *url_signed_str;
char *bucket_name = "<your bucket name>";
char * object_name = "<your object name>";
aos_string_t url;
aos_string_t local_file;
char *filename = "<local filename>"

aos_pool_create(&p, NULL);
// init_ oss_request_options
...

headers = aos_table_make(p, 0);
resp_headers = aos_table_make(p, 5);
req = aos_http_request_create(p);
req->method = HTTP_PUT;
url_str = gen_test_signed_url(oss_request_options,bucket_name, object_name, expire_time, req);
aos_str_set(&url, url_str);
```

```
aos_str_set(&local_file, filename);
s = oss_put_object_from_file_by_url(oss_request_options, &url, &local_file, headers, &resp_headers);

aos_pool_destroy(p);
```

Error Responses

Common Error Codes

Error Code	Description
AccessDenied	Access denied
BucketAlreadyExists	The bucket already exists
BucketNotEmpty	The bucket is not empty
EntityTooLarge	The entity is too large
EntityTooSmall	The entity is too small
FileGroupTooLarge	The file group is too large
FilePartNotExist	A file part does not exist
FilePartStale	A file part has expired
InvalidArgument	Parameter format error
InvalidAccessKeyId	The Access Key ID does not exist
InvalidBucketName	The bucket name is invalid
InvalidDigest	The digest is invalid
InvalidObjectName	The object name is invalid
InvalidPart	A part is invalid
InvalidPartOrder	The part order is invalid
InvalidTargetBucketForLogging	The logging operation has an invalid target bucket
InternalError	Internal OSS error
MalformedXML	Illegal XML format
MethodNotAllowed	The method is not supported
MissingArgument	A parameter is missing
MissingContentLength	The content length is missing
NoSuchBucket	The bucket does not exist
NoSuchKey	The file does not exist

NoSuchUpload	Multipart Upload ID does not exist
NotImplemented	The method cannot be processed
PreconditionFailed	Preprocessing error
RequestTimeTooSkewed	The request initiation time exceeds the server time by 15 minutes
RequestTimeout	Request timed out
SignatureDoesNotMatch	Signature error
TooManyBuckets	The user's bucket quantity exceeds the limit

Download SDK

Java SDK

Java SDK Documentation

Click to View

Java SDK (2015-07-10) Version 2.0.5

Java SDK download address:java_sdk_20150710.zip

Updates:

- Added Append Object support.
- Added HTTPS support.
- Added the encoding-type parameter in the DeleteObject and ListObjects interfaces for users to specify the object name encoding method.
- Removed the mandatory check of the Expires response header date format, fixing the issue where the system could not parse Expires response headers without using the GMT date format.

Java SDK (2015-05-29) Version 2.0.4

Java SDK download address:java_sdk_20150529.zip

Updates:

- Added STS support.
- Added a test Demo Jar; for specific usage, refer to Readme.txt in the demo folder.
- Modified the CreateBucket logic to allow users to specify the Bucket ACL during creation.
- Modified the bucket name inspection rules, allow for operations on existing buckets with underlines, but not the creation of buckets with underlines.
- Optimized the HTTP connection pool, enhancing its concurrent processing capability.

Java SDK (2015-04-24) Version 2.0.3

Java SDK download address:java_sdk_20150424.zip

Updates:

- Added the deleteObjects interface to batch delete object lists.
- Added the doesObjectExist interface to check for the existence of objects under the specified bucket.
- Added the switchCredentials interface, allowing users to switch the credentials of existing OSSClient instances.
- Added the OSSClient constructor with CredentialsProvider to allow for more customization of the CredentialsProvider.
- Fixed the bug that occurred when the Source Key in the copyObject interface contained the plus sign special character.
- Adjusted the OSSException/ClientException information display format.

Java SDK (2015-03-23) Version 2.0.2

Java SDK download address:java_sdk_20150323.zip

Updates:

- Added the GeneratePostPolicy/calculatePostSignature interface, used to generate the Policy string and Post signature for Post requests.
- Support for Bucket Lifecycle.
- Added the URL signature-based Put/GetObject overload interface.
- Support for PutObject, UploadPart, and Chunked encoding data upload.
- Fixed several bugs.

Java SDK (2015-01-15) Version 2.0.1

Java SDK download address:java_sdk_20150115.zip

Updates:

- Support for Cname, allowing users to specify which domain names are retained
- Support for user ContentType and ContentMD5 specification during URI generation.
- Addressed the problem where CopyObject requests did not support server-side encryption.
- Changed the UserAgent format.
- Expanded location constant and added some samples

Java SDK (2014-11-13)

Java SDK download address:java_sdk_20141113.zip

New content: Upload Part Copy function OSS Java SDK source code Sample Code

Important: In version 2.0.0, the OSS Java SDK removed the OTS-related code from previous versions, adjusted the package structure, and added OSS SDK source code and sample code. Programs developed using a version earlier than 2.0.0 must change the referenced package name when using version 2.0.0. The package names com.aliyun.openservices.* and com.aliyun.openservices.oss.* should be changed to com.aliyun.oss.*.

Python SDK

Python SDK Documentation

Click to View

Python SDK (2015-09-09) Version 0.4.2

Updates:

- Added the appendfromfile interface in osscmd to support automatically appending file content to a specified object
- Added apengd (append object interface) in API

Python SDK download address: OSS_Python_API_20150909.zip

Python cmd tool instructions for use: Click to View

Python SDK (2015-08-11) Version 0.4.1

Updates:

- osscmd supports the receipt of xml encoded responses that contain control character lists

and delete objects.

Python SDK download address: OSS_Python_API_20150811.zip

Python cmd tool instructions for use: Click to View

Python SDK (2015-07-07) Version 0.4.0

Updates:

- Supported STS function in osscmd

Python SDK download address: OSS_Python_API_20150707.zip

Python cmd tool instructions for use: Click to View

Python SDK (2015-06-24) Version 0.3.9

Updates:

- Added copylargefile command in osscmd to support the copying of large files

Python SDK download address: OSS_Python_API_20150624.zip

Python cmd tool instructions for use: Click to View

Python SDK (2015-04-13) Version 0.3.8

Updates:

- Fixed the invalid max_part_num specified for multiupload problem in osscmd
- Added an md5 check for the part specified by upload_part in oss_api

Python SDK download address: oss_python_sdk_20150413.zip

Python cmd tool instructions for use: Click to View

Python SDK (2015-01-29) Version 0.3.7

Updates:

- Added the referer and lifecycle interfaces in oss_api.
- Added referer and lifecycle commands in osscmd.
- Fixed invalid upload_id in osscmd.

Python SDK download address: oss_python_sdk_20150129.zip

Python cmd tool instructions for use: Click to View

Python SDK (2014-12-31) Version 0.3.6

Updates:

- Added the check_point function for the osscmd uploadfromdir command, using the --check_point optional setting.
- Added the --force function for the osscmd deleteallobject command, force deleting all files.
- Added the --thread_num option in osscmd's multipart and uploadfromdir/downloadtodir commands, allowing users to adjust the number of threads
- Added the file name-based Content-Type generation function in oss_api.
- Added the --temp_dir option for the osscmd downloadtodir command, supporting temporarily saving the downloaded file to the specified directory.
- Added the --check_md5 option for osscmd, allowing md5 checks on upload files.

Python SDK download address: oss_python_sdk_20141231.zip

For a Quick Start Guide, refer to the README file in the SDK

Python SDK (2014-05-09)

Updates:

- Fixed the bug of logger initialization error in oss_util.
- Optimized the multi_upload_file upload interface in oss_api in certain situations, reducing the number of re-uploads due to network exceptions.

Python SDK download address:oss_python_sdk_20140509.zip

Android SDK

Android SDK Documentation (for version 1.2.0 in particular)

Click to View

Android SDK (07-31-2015) Version 1.3.0

Address for SDK download:OSS_Android_SDK_20150731

Updates:

1. Released an independent multipart upload interface.
2. Added the OSSData direct import from input stream interface.
3. Fixed the problem in STS mode where a single token was retrieved multiple times upon initialization.
4. Fixed the problem where, after a breakpoint download is manually canceled, the stream was not fully read, affecting connection multiplexing.

Android SDK (07-01-2015) Version 1.2.0

Address for SDK download:OSS_Android_SDK_20150701

Updates:

1. Uses httpdns for domain name resolution to prevent domain name hijacking.
2. Optimized connection multiplexing to increase stability in concurrent requests.

Android SDK (06-02-2015) Version 1.1.0

Address for SDK download:OSS_Android_SDK_20150602

Updates:

1. Added support for STS authentication.
2. Added commonPrefixs in the ListObjectsInBucket results.
3. Added the method of retrieving an object input stream.

Android SDK (04-07-2015) Version 1.0.0

Address for SDK download:OSS_Android_SDK_20150407

> Note that this OSS Android SDK has been formally integrated into the AliCloud OneSDK. For a uniform style, the SDK name and several interface names were changed in this update. For details, see the documentation.

Updates:

1. Support for ListObjectsInBucket.
2. Support for breakpoint downloads.
3. Support for global network parameter settings and resumable data transfer configuration options.
4. Unified the style of the package name with AliCloud OneSDK: Changed the package name from 'com.aliyun.mbaas.oss.*' to 'com.alibaba.sdk.android.oss.*'.

Android SDK Documentation (for version 0.3.0 in particular)

Click to Download

Android SDK (01-23-2015) Version 0.3.0

Address for SDK download:OSS_Android_SDK_20150123

Updates:

1. Improved support for CNAME and CDN domain names
2. Added the custom benchmark time interface
3. In asynchronous operations, tokens are generated in sub-threads
4. Checks HTTP exception responses to see if they are from the OSS Server

Android SDK (12-20-2014) Version 0.2.2

Address for SDK download:OSS_Android_SDK_20141220

ChangeList:

1. Fixed the bug in version 0.2.1 where the incorrect BucketName was entered in the objectKey parameter of the asynchronous upload interface

Android SDK (12-17-2014) Version 0.2.1

Address for SDK download:OSS_Android_SDK_20141217

Updates:

1. Added the OSSBucket class to set domain names, permissions and signatures for individual buckets.
2. Added the permission setting to specify access permissions for a bucket.
3. Added the objectKey parameter for asynchronous task progress and exception callbacks.
4. Access URLs can be generated for one OSSObject to facilitate access to URLs by third-parties.
5. All asynchronous upload/download tasks can be canceled midway through.
6. Fixed the breakpoint upload interface's invalid Content-type bug.

Android SDK (11-26-2014) Version 0.0.1

Address for SDK download:OSS_Android_SDK_20141126

iOS SDK

iOS SDK Documentation (for version 1.3.0 in particular)

Click to View

iOS SDK (2015-08-05) Version 1.3.0

Address for SDK download:OSS_iOS_SDK_20150805

Updates:

1. Released an independent multipart upload interface
2. In STS authentication mode, the SDK automatically manages a token's lifecycle and gets a new one only after expiration
3. Fixed the problem where concurrent upload tasks could not always be canceled
4. Asynchronous upload/download interfaces return a handler, through which tasks are canceled
5. Added a complete demo
6. Supports the servercallback function

iOS SDK (2015-06-30) Version 1.2.0

Address for SDK download:OSS_iOS_SDK_20150630

Updates:

1. Uses httpdns for domain name resolution to prevent domain name hijacking

iOS SDK (2015-06-09) Version 1.1.0

Address for SDK download:OSS_iOS_SDK_20150609

Updates:

1. Changed the method of using SDK from the original .an object to framework
2. Added STS support
3. Fixed the bug of invalid callback in some situations

iOS SDK (2015-04-07) Version 1.0.0

Address for SDK download:OSS_iOS_SDK_20150407

> Note: This OSS iOS SDK has been formally integrated into the AliCloud OneSDK. For a uniform style, the SDK package name and several interface names were changed in this update. For details, see the SDK documentation

Updates:

1. Added the list Objects function
2. Enabled the function of specified range download to define the end of an object
3. Added a new method of using OSS SDK

iOS SDK Documentation (for version 0.1.2 in particular)

Click to Download

iOS SDK (2015-03-04) Version 0.1.2

Address for SDK download:OSS_iOS_SDK_20150304

Updates:

1. Supports the upload of object keys with names including Chinese characters
2. Fixed the bugs of the resumable data transfer function

iOS SDK (2015-01-20) Version 0.1.1

Address for SDK download:OSS_iOS_SDK_20150120

Updates:

1. Changed the methods provided by the SDK from the original framework to static library
2. The SDK simultaneously provides three types of static libraries: real machine, simulator, and joint real machine and simulator use
3. Added bucket settings to direct to the domain name interface of the bound CNAME

iOS SDK (2014-12-22) Version 0.1.0

Address for SDK download:OSS_iOS_SDK_20141222

PHP SDK

PHP SDK (2015-08-19)

Download address:oss_php_sdk_20150819

Updates

- Fixed the download signature mismatch problem when using response-content-disposition and other HTTP headers.
- Added settings for response body conversion. The OSS currently supports xml, array, and json formats. XML is the default format
- Added the copy_upload_part method
- Added support for STS
- Changed the $options parameter location in the signature URL
- Fixed the read_dir looping problem
- Added the referer and lifecycle interfaces.Added the content-md5 check option for upload by file and multipart upload.
- Added init_multipart_upload to directly obtain string type uploads
- Adjusted the return value of the batch_upload_file function from the original blank value to a boolean value; true indicates success and false indicates failure.
- Adjusted the tool function location in tsdk.class.php, placing it in util/oss_util.class.php. If you need to reference it, add OSSUtil:: and reference this file.

Bug fixes:

- Fixed the problem in the Copy object process where you could not edit the header.
- Fixed the custom upload syntax error during upload part.
- Fixed the problem where the mimetype of office2007 files could not be set correctly during uploads.
- Fixed the problem where the system would time out and quit when it encountered an empty directory during the batch_upload_file operation.

PHP SDK (2014-06-25)

Download address:oss_php_sdk_20140625 New functions:

- Added the CORS setting function

PHP SDK V1 (2013-06-25)

PHP SDK (2012-10-10)

This version mainly includes a change to the domain name generation rules according to the newly

released API

PHP SDK (2012-08-17)

This version is primarily designed to solve the problem where fix get_sign_url could not set Expires

PHP SDK (2012-06-12)

Updates:

1. Fixed the bugs in hostname setting
2. Optimized internal exception handling
3. Added support for third-level domain names, e.g. bucket.storage.aliyun.com
4. Optimized the demo program to make it simpler

C SDK

OSS C SDK Documentation

Click to View

OSS C SDK (2015-08-17) Version 0.04

Download address:

- Linux:aliyun_OSS_C_SDK_v0.04.tar.gz
- Windows:aliyun_OSS_C_SDK_windows_v0.0.4.rar

Updates:

1. Supports keeplive persistent connections
2. Supports lifecycle settings

OSS C SDK (2015-07-08) Version 0.03

Download address:

- Linux:aliyun_OSS_C_SDK_v0.03.tar.gz
- Windows:aliyun_OSS_C_SDK_windows_v0.0.3.rar

Updates:

1. Added oss_append_object_from_buffer interface, allowing the user to append buffer content to the object
2. Added oss_append_object_from_file interface, allowing the user to append file content to the object

OSS C SDK (2015-06-10) Version 0.0.2

Updates:

1. Added the oss_upload_part_copy interface to support the Upload Part Copy method
2. Enabled temporary access to OSS through temporary authorization from STS service

OSS C SDK (2015-05-28) Version 0.0.1

Updates:

1. Added the oss_create_bucket interface to create OSS buckets
2. Added the oss_delete_bucket interface to delete OSS buckets
3. Added the oss_get_bucket interface to get OSS buckets' ACLs
4. Added the oss_list_object interface to list objects in OSS buckets
5. Added oss_put_object_from_buffer interface, allowing the user to upload buffer content to the object
6. Added oss_put_object_from_file interface, allowing the user to upload file content to the object
7. Added oss_get_object_to_buffer interface, allowing the user to download object content to the buffer
8. Added oss_get_object_to_file interface, allowing the user to download object content to the file
9. Added the oss_head_object interface, to get objects' user meta information
10. Added the oss_delete_object interface to delete objects
11. Added the oss_copy_object interface to copy objects
12. Added the oss_init_multipart_upload interface to initialize multipart uploads
13. Added oss_upload_part_from_buffer interface, allowing the user to upload buffer content to the part
14. Added oss_upload_part_from_file interface, allowing the user to upload file content to the part
15. Added the oss_list_upload_part interface, which retrieves information for all uploaded parts
16. Added the oss_complete_multipart_upload interface for multipart upload
17. Added the oss_abort_multipart_upload interface to cancel multipart upload events
18. Added the oss_list_multipart_upload interface to get all multipart upload events in the bucket
19. Added the oss_gen_signed_url interface to generate a signed URL
20. Added the oss_put_object_from_buffer_by_url interface to upload the buffer content to the

object using the URL signature method
21. Added the oss_put_object_from_file_by_url interface to upload the file content to the object using the URL signature method
22. Added the oss_get_object_to_buffer_by_ur interface download the object content to the buffer using the URL signature method
23. Added the oss_get_object_to_file_by_ur interface to download the object content to the file using the URL signature method
24. Added the oss_head_object_by_url interface to get objects' user meta information using the URL signature method

OSS NodeJs SDK

OSS NodeJs SDK

The link to github is as follows: NodeJs SDK

.NET SDK

.NET SDK Documentation

Click to View

.NET SDK (05-28-2015)

Download address:aliyun_dotnet_sdk_20150528

Updates:

2015/05/28

- Added Bucket Lifecycle support. Adding and removing Lifecycle rules allowed;
- Added the DoesBucketExist and DoesObjectExist interfaces for determining the existence of Bucket and Objects;
- Added SwitchCredentials so as to be able to change user account information at runtime;
- Added the ICredentialsProvider interface class to provide a policy for generating customized Credentials through its implementation.
- Added GeneratePostPolicy interface to generate Post Policy ;
- Added asynchronization interface (supporting Put/Get/List/Copy/PartCopy asynchronous

operations);
- Added STS support.
- Added custom time calibration function. It can be set through the Client configuration item - SetCustomEpochTicks interface;
- Added support for Chunked encoding transfer. The Content-Length item can be skipped when uploading;
- Fixed the bug of getting null results after setting the Expose Header attribute in Bucket CORS;
- Fixed the bug of SDK only getting the first Prefix out of multiple prefixes contained in CommonPrefixs returned in an ListObjects request.
- Fixed the bug of RequestId and HostId resolved to null in response to an OSS-related exception;
- Fixed the bug of encoding error due to the inclusion of Chinese characters in the source key of the CopyObject/CopyPart interface;

.NET SDK (01-15-2015)

Download address:aliyun_dotnet_sdk_20150115

Environment requirements:

- .NET Framework 4.0 and above
- A registered user account on AliCloud.com

Assembly:Aliyun.OSS.dll

Version:1.0.5492.31618

Package structure:

- bin
  - Aliyun.OSS.dll .NET assembly file
  - Aliyun.OSS.pdb debugging and project status information file
  - Aliyun.OSS.xml Assembly comments file
- doc
  - Aliyun.OSS.chm Help file
- src
  - SDK source code
- sample
  - Sample code

Updates:

- Removed OTS branch, assembly renamed to Aliyun.OSS.dll
- .NET Framework version upgraded to 4.0 and above
- OSS: Added interfaces for Copy Part, Delete Objects, Bucket Referer List, etc.

- OSS: Added the ListBuckets pagination function
- OSS: Added CNAME support
- OSS: Fixed the Put/GetObject flow interruption problem
- OSS: Added samples

.NET SDK (06-26-2014)

Download address:aliyun_dotnet_sdk_20140626

Open Services SDK for .NET included OSS and OTS SDKs..NETSDK used the same interface design as Java SDK and made some improvements based on C#. (The latest version supports multipart uploads to the OSS)

Environment requirements:

- .NET Framework 3.5 SP1 or above
- A registered user account on AliCloud.com, and a subscription to related services (OSS, OTS).

Updates: 2014/06/26- OSS:

- Added CORS functionality.

2013/09/02- OSS:

- Fixed the bug of being unable to throw correct exceptions in some cases.
- Optimized SDK performance.

2013/06/04- OSS:

- Changed the default OSS service access method to a third-level domain name access method.

2013/05/20- OTS:

- Updated the default OTS service address to:http://ots.aliyuncs.com
- Added Mono support.
- Fixed some bugs in the SDK, so that it runs more stably.

2013/04/10- OSS:

- Added Object Multipart Upload functionality.
- Added Copy Object functionality.
- Added the ability to generate pre-signed URLs.
- Isolated the IOSS interface to be inherited by OssClient.

2012/10/10- OSS:

- Updated the default OSS service address to:http://oss.aliyuncs.com

2012/09/05- OSS:

- Resolved the problem of invalid parameters like Prefix for ListObjects.

2012/06/15- OSS:

- Added OSS support for the first time.Included basic operations, such as create, modify, read and delete, on OSS Bucket, ACL and Object.
- OTS:
- OTSClient.GetRowsByOffset supports reverse read.
- Added an automatic error handling mechanism for special requests.
- Added HTML help files.

2012/05/16- OTS

- Client.GetRowsByRange supports reverse read.

2012/03/16- OTS

- Access interface, including table/table group creation, modification, and deletion, as well as data insertion, modification, deletion, query, etc.
- Client access settings, such as proxy and HTTP connection attribute settings
- Unified the structure for exception handling.