

**Video on Demand**

# **SDK Reference**

**Issue**            01  
**Date**             2026-03-27



**Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2026. All rights reserved.**

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Cloud Computing Technologies Co., Ltd.

## **Trademarks and Permissions**



HUAWEI and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

## **Notice**

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

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

## **Huawei Cloud Computing Technologies Co., Ltd.**

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

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

---

# Contents

---

<b>1 SDK Overview.....</b>	<b>1</b>
<b>2 SDK Download.....</b>	<b>2</b>
<b>3 SDK Development Description.....</b>	<b>3</b>

# 1 SDK Overview

## VOD SDK Overview

The VOD SDK encapsulates VOD API requests. Before using the SDK, read the VOD API document to understand the functions, parameters, rules, and usage methods.

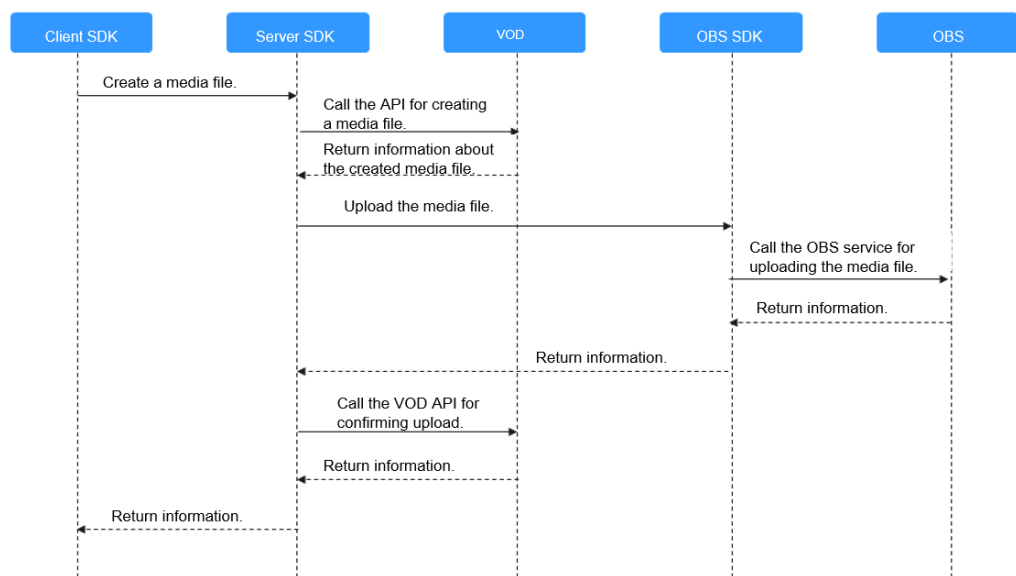
The SDK 2.x has been brought offline. Now, the newly designed SDK V3 has been launched. The latest SDK supports **Java, Python, Go, Node.js, .Net, PHP, and C++**. You can log in to [SDK Center](#) to download the desired SDK.

## Process of Integrating SDK for Development



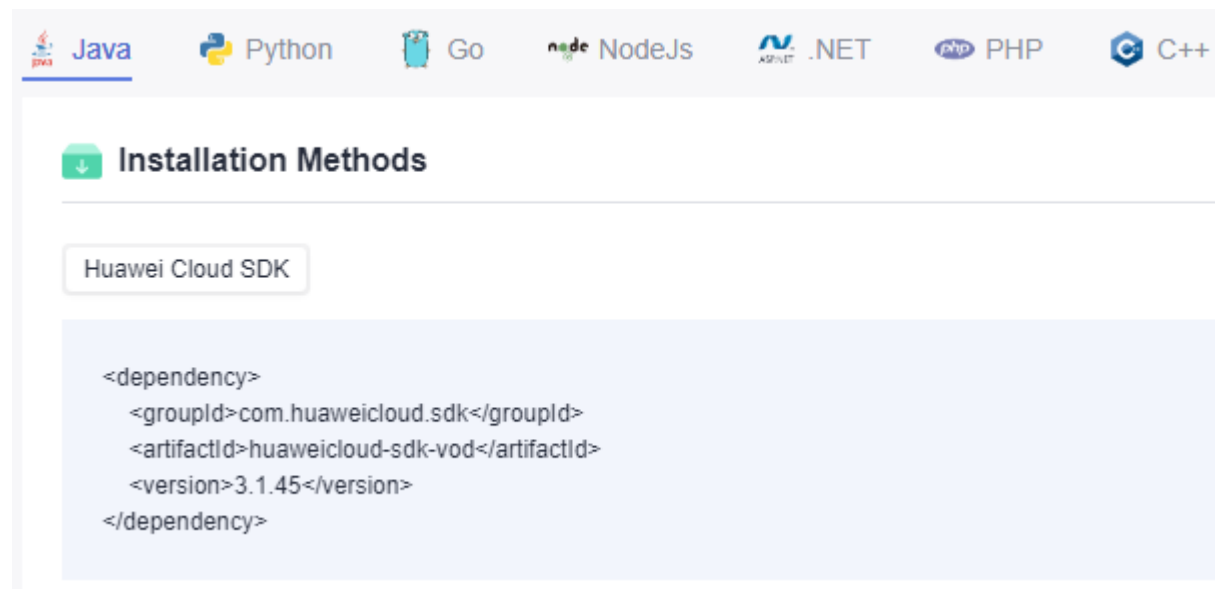
## SDK Working Process

In addition to the VOD SDK, the server SDK interacts with other SDKs as follows.



# 2 SDK Download

Huawei Cloud VOD provides server SDKs in **Java**, **Python**, **Go**, **Node.js**, **.NET**, **PHP**, and **C++**. You can access the [SDK Center](#) to download the desired SDK.



The screenshot displays the Huawei Cloud SDK Center interface. At the top, there is a navigation bar with icons and labels for various programming languages: Java, Python, Go, Node.js, .NET, PHP, and C++. Below the navigation bar, the 'Installation Methods' section is visible, featuring a green download icon and the title 'Installation Methods'. Underneath, there is a button labeled 'Huawei Cloud SDK'. The main content area shows a code block with the following Maven dependency XML:

```
<dependency>
  <groupId>com.huaweicloud.sdk</groupId>
  <artifactId>huaweicloud-sdk-vod</artifactId>
  <version>3.1.45</version>
</dependency>
```

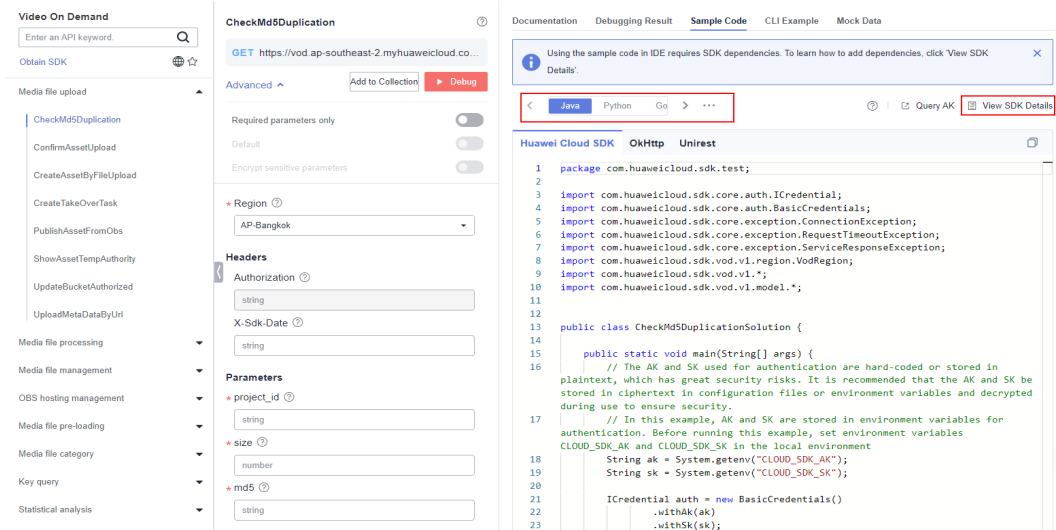
# 3 SDK Development Description

The SDK V3 encapsulates all APIs provided by VOD. You can use the [API Explorer](#) to debug the APIs. Demos in different programming languages are generated for your reference.

## NOTE

If you have any questions or suggestions when using SDK V3, [submit a service ticket](#) to give feedback.

Figure 3-1 API Explorer



## Uploading Media Files

When you need to use the server SDK to upload local media files, perform operations by referring to [application example 1](#) or [application example 2](#) in *VOD API Reference*. For details about how to upload a media file and confirm file upload in the application example, see [Uploading Media Files to VOD](#) and [Verifying the Upload](#) in the SDK. You can use HTTP PUT to upload a media file to the corresponding URL.

## Generating a Signed URL

When using the SDK, you can use the [URL validation](#) algorithm on the VOD console or refer to the following demo to generate a signed URL.

```
package AuthUrlDemo;
import org.apache.commons.codec.binary.Hex;
import org.apache.commons.codec.digest.DigestUtils;
import org.apache.commons.lang3.StringUtils;

import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;

import java.net.URL;
import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;
import java.text.SimpleDateFormat;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.*;

class AuthUrlDemo {
    // url is the original playback URL that is not encrypted. key is the key value configured on the VOD console.
    // Encryption algorithm A
    public static String createAuthInfoUrlByAlgorithmA(String url, String key, String method) {
        try {
            checkParam(url, key);

            long timestamp = Instant.now().getEpochSecond();
            String randUid = UUID.randomUUID().toString().replaceAll("-", "");
            String uid = "0";
            String tmpRandKey = timestamp + "-" + randUid + "-" + uid;

            URL originUrl = new URL(url);
            String originHashStr = originUrl.getPath() + "-" + tmpRandKey + "-" + key;
            String hashStr = hashString(originHashStr, method);
            String authInfo = "auth_key=" + tmpRandKey + "-" + hashStr;

            return StringUtils.isEmpty(originUrl.getQuery()) ? url + "?" + authInfo : url + "&" + authInfo;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
    // Encryption algorithm B
    public static String createAuthInfoUrlByAlgorithmB(String url, String key, String method) {
        try {
            checkParam(url, key);

            URL originUrl = new URL(url);
            String filePath = originUrl.getPath();
            String dateStr = LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyyMMddHHmm"));
            String originHashStr = key + dateStr + filePath;
            String hashStr = hashString(originHashStr, method);

            return originUrl.getProtocol() + "://" + originUrl.getHost() + "/"
                + dateStr + "/" + hashStr + originUrl.getFile();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

```
// Encryption algorithm C
public static String createAuthInfoUrlByAlgorithmC(String url, String key, String method) {
    try {
        checkParam(url, key);

        URL originUrl = new URL(url);
        String filePath = originUrl.getPath();
        String hexTime = Long.toHexString(Instant.now().getEpochSecond()).toUpperCase(Locale.ENGLISH);
        String originHashStr = key + filePath + hexTime;
        String hashStr = hashString(originHashStr, method);

        return originUrl.getProtocol() + "://" + originUrl.getHost() + "/" + hashStr + "/" + hexTime +
originUrl.getFile();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}

// Encryption algorithm D
public static String createAuthInfoUrlByAlgorithmD(String url, String key, String pliveDuration, String
previewDuration) {
    try {
        isDigital("Preview duration", previewDuration);
        isDigital("Pseudo-streaming start time", pliveDuration);
        checkParam(url, key, previewDuration, pliveDuration);

        String authArg = "";

        URL originUrl = new URL(url);
        String urlPath = originUrl.getPath();
        String pathInUrl = urlPath.substring(0, urlPath.lastIndexOf("/") + 1);
        String data = encodeUrl(pathInUrl) + "$" + getUtcTime();

        if (previewDuration != null && !previewDuration.isEmpty()) {
            data += "$" + previewDuration;
            authArg = "&exper=" + previewDuration;
        }

        if (pliveDuration != null && !pliveDuration.isEmpty()) {
            data += "$" + pliveDuration;
            authArg = "&plive=" + pliveDuration;
        }
        String encryptInfo = aesCbcEncrypt(data, key, true);

        String authInfoStr = "auth_info=" + URLEncoder.encode(encryptInfo, StandardCharsets.UTF_8);
        if (url.contains("?")) {
            return originUrl + "&" + authInfoStr + authArg;
        }
        return originUrl + "?" + authInfoStr + authArg;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}

public static String createAuthInfoUrlByAlgorithmE(String url, String key, String pliveDuration, String
previewDuration) {
    try {
        isDigital("Preview duration", previewDuration);
        isDigital("Pseudo-streaming start time", pliveDuration);
        checkParam(url, key, previewDuration, pliveDuration);

        URL originUrl = new URL(url);
        String urlPath = originUrl.getPath();

        String currentTimestamp = String.valueOf(new Date().getTime() / 1000);

        String authArg = "";
        String authStr = key + urlPath + currentTimestamp;
```

```
        if (previewDuration != null && !previewDuration.isEmpty()) {
            authArg = "&exper=" + previewDuration;
            authStr = authStr + previewDuration;
        }

        if (pliveDuration != null && !pliveDuration.isEmpty()) {
            authArg = "&plive=" + pliveDuration + authArg;
            authStr = authStr + pliveDuration;
        }

        authStr = sha256DigestAsHex(authStr);
        authArg = "auth_key=" + authStr + "&timestamp=" + currentTimestamp + authArg;

        return url.contains("?") ? originUrl + "&" + authArg : originUrl + "?" + authArg;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}

private static void checkParam(String url, String key) {
    if (StringUtils.isEmpty(url, key)) {
        throw new IllegalArgumentException("url or key is illegal");
    }
}

private static void checkParam(String url, String key, String preview, String plive) {
    if (StringUtils.isEmpty(url, key)) {
        throw new IllegalArgumentException("url or key is illegal");
    }

    if (StringUtils.isNotEmpty(preview, plive)) {
        throw new IllegalArgumentException("preview and plive cannot be enabled all ");
    }
}

private static String aesCbcEncrypt(String data, String key, boolean hasPoint) throws Exception {
    checkParam(data, key);

    byte[] realKey = get128BitKey(key);
    SecureRandom secureRand = new SecureRandom();
    byte[] ivBytes = new byte[16];
    secureRand.nextBytes(ivBytes);

    if (hasPoint) {
        return aesCbcEncrypt(data, ivBytes, realKey) + "." + bytesToHexString(ivBytes);
    } else {
        return aesCbcEncrypt(data, ivBytes, realKey) + bytesToHexString(ivBytes);
    }
}

private static String aesCbcEncrypt(String data, byte[] ivBytes, byte[] key) throws Exception {
    SecretKeySpec sk = new SecretKeySpec(key, "AES");
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

    if (ivBytes != null) {
        cipher.init(Cipher.ENCRYPT_MODE, sk, new IvParameterSpec(ivBytes));
    } else {
        cipher.init(Cipher.ENCRYPT_MODE, sk);
    }

    return Base64.getEncoder().encodeToString(cipher.doFinal(data.getBytes(StandardCharsets.UTF_8)));
}

private static byte[] get128BitKey(String key) {
    byte[] result = null;
}
```

```
    if (key != null) {
        result = new byte[16];
        byte[] origin = key.getBytes();

        System.arraycopy(origin, 0, result, 0, Math.min(origin.length, 16));
    }

    return result;
}

private static String encodeUrl(String str) {
    try {
        if (StringUtils.isNotEmpty(str)) {
            StringBuilder encodeStr = new StringBuilder(32);
            String[] tmpArray = str.split("/");
            for (String s : tmpArray) {
                encodeStr.append(URLEncoder.encode(s, StandardCharsets.UTF_8)).append("/");
            }
            return encodeStr.toString();
        }
    } catch (Exception e) {
        throw new RuntimeException(String.format("Encode fail %s", e.getMessage()));
    }
    return str;
}

public static String sha256DigestAsHex(String plainText) {
    MessageDigest messageDigest;
    String encodeStr;
    try {
        messageDigest = MessageDigest.getInstance("SHA-256");
        byte[] hash = messageDigest.digest(plainText.getBytes(StandardCharsets.UTF_8));
        encodeStr = Hex.encodeHexString(hash);
    } catch (NoSuchAlgorithmException e) {
        throw new IllegalStateException("Could not find MessageDigest with algorithm SHA-256", e);
    }
    return encodeStr;
}

private static String getUtcTime() {
    SimpleDateFormat foo = new SimpleDateFormat("yyyyMMddHHmmss");
    java.util.Calendar cal = java.util.Calendar.getInstance();
    int zoneOffset = cal.get(java.util.Calendar.ZONE_OFFSET);
    int dstOffset = cal.get(java.util.Calendar.DST_OFFSET);
    cal.add(java.util.Calendar.MILLISECOND, -(zoneOffset + dstOffset));

    return foo.format(new Date(cal.getTimeInMillis()));
}

private static String bytesToHexString(byte[] src) {
    StringBuilder stringBuilder = new StringBuilder();
    if (src == null || src.length == 0) {
        return null;
    }
    for (byte b : src) {
        int v = b & 0xFF;
        String hv = Integer.toHexString(v);
        if (hv.length() < 2) {
            stringBuilder.append(0);
        }
        stringBuilder.append(hv);
    }
    return stringBuilder.toString();
}

private static void isDigital(String paramName, String value) throws Exception {
    if (value != null && !value.isEmpty() && !value.matches("\\d+")) {
        throw new Exception (paramName + "Only digits are allowed.");
    }
}
```

```
}  
private static String hashString(String oriStr, String algorithm) {  
    return algorithm.equals("MD5") ? DigestUtils.md5Hex(oriStr.getBytes(StandardCharsets.UTF_8))  
        : sha256DigestAsHex(oriStr);  
}  
}
```