# Ant Technology

Mobile Sync Service

User Guide

Document Version: 20260303

蚂蚁集团
ANT GROUP

# Legal disclaimer

# Document conventions

| Style | Description | Example |
|---|---|---|
| ⚠ **Danger** | A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results. | ⚠ **Danger:** Resetting will result in the loss of user configuration data. |
| 🔔 **Warning** | A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results. | 🔔 **Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance. |
| 📢 **Notice** | A caution notice indicates warning information, supplementary instructions, and other content that the user must understand. | 📢 **Notice:** If the weight is set to 0, the server no longer receives new requests. |
| ? **Note** | A note indicates supplemental instructions, best practices, tips, and other content. | ? **Note:** You can use Ctrl + A to select all files. |
| > | Closing angle brackets are used to indicate a multi-level menu cascade. | Click **Settings**> **Network**> **Set network type**. |
| **Bold** | Bold formatting is used for buttons , menus, page names, and other UI elements. | Click **OK**. |
| `Courier font` | Courier font is used for commands | Run the `cd /d C:/window` command to enter the Windows system folder. |
| *Italic* | Italic formatting is used for parameters and variables. | `bae log list --instanceid` *Instance_ID* |
| [] or [a\|b] | This format is used for an optional value, where only one item can be selected. | `ipconfig [-all\|-t]` |
| {} or {a\|b} | This format is used for a required value, where only one item can be selected. | `switch {active\|stand}` |

# Table of Contents

1.Change history ---- 05
2.About Mobile Sync Service ---- 06
3.Terms ---- 09
4.Client-side development ---- 11
4.1. Integrate with Android ---- 11
4.2. Integrate with iOS ---- 13
4.2.1. Add SDK ---- 13
4.2.2. Use SDK ---- 14
5.Server-side Integration ---- 19
5.1. Server-side integration guide ---- 19
5.2. Connect using the Java SDK ---- 19
5.3. User consistency validation ---- 27
6.Console operations ---- 30
6.1. Console overview ---- 30
6.2. Add configuration ---- 30
6.3. Send business data ---- 32
6.4. View configuration details ---- 34
6.5. Modify configuration ---- 34
6.6. Offline configuration ---- 35
6.7. Query statistics for configuration pushes ---- 36
6.8. Service management ---- 36
6.9. Query MSS operation records ---- 36
7.API reference ---- 38
7.1. Android API ---- 38
7.2. iOS API ---- 43

# 1.Change history

| Document version | Revisions |
|---|---|
| V20251215 | • Merge the Android, iOS, and HarmonyOS console documentation.<br>• Enable the operation log feature for the console on Private Cloud and International Site. |
| V20210630 | Added the Query data synchronization history section, which provides instructions on how to view synchronization records. |

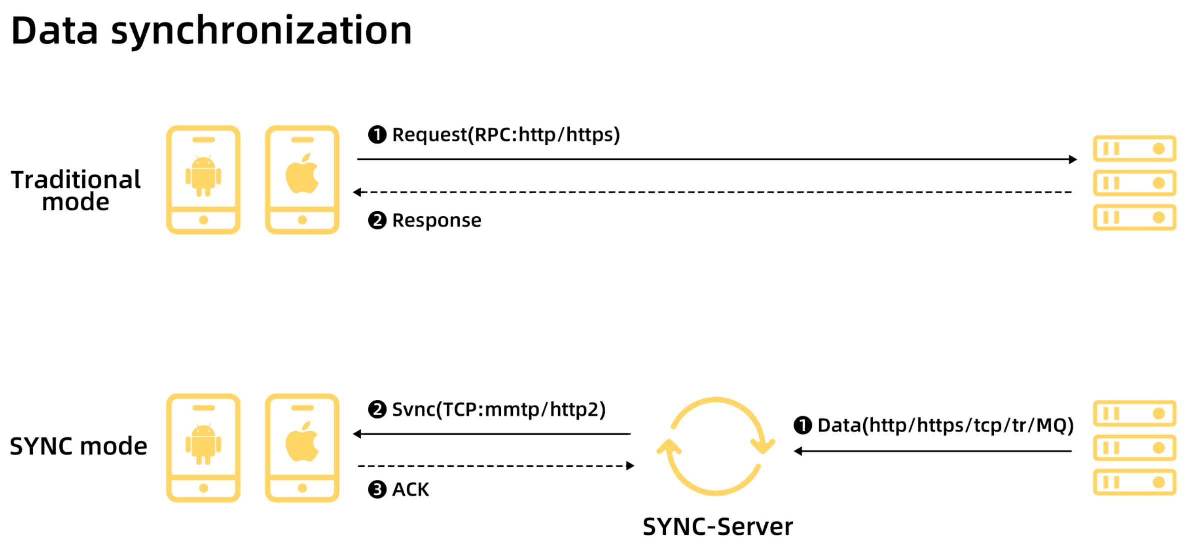
# 2.About Mobile Sync Service

Mobile Sync Service (MSS) is a core basic business component of the mPaaS platform. MSS originates from the E2E solution SYNC of Ant Financial Group, which is oriented to mobile Apps and pushes massive data from the server to the client. This component provides a secure data channel based on the Transmission Control Protocol (TCP) and Secure Sockets Layer (SSL). This data channel can actively synchronize business data from the server to the client App on a timely, accurate, and orderly manner.

Traditional RPC has been applied in the Internet industry for decades and can meet most business scenarios and functional requirements. However, the popularization and development of the mobile Internet have driven the App scale and users’ requirements for Apps to a new stage. Traditional RPC requests have many drawbacks due to their own characteristics.

- In certain scenarios, a client needs to call RPC requests to obtain the latest data, but actually no or only little data on the server (cloud) changes.
- As different business modules and functions are designed to be independent of each other, they need to call RPC requests respectively to obtain their business data when the client starts.
- The client cannot be promptly aware of the data changes on the server but needs to call RPC APIs in polling mode to update data.
- Traditional RPC performs data interactions mostly based on HTTP(S) short connections. This type of connection cannot be persistent even if by using features such as keep-alive. In other words, links cannot be reused continuously. Requests for connection creation, certificate exchange, and encryption/decryption will increase the time consumption and compromise the network performance.

MSS is introduced to improve or solve these problems.



## Features

The core features of MSS are described as follows:

- Reliable synchronization

  For business scenarios where the quality of service (QoS) level is arrival guarantee, MSS ensures that the data pushed from your server will be certainly synchronized to the client if the user is active within the data validity period and meets the push requirements of your server.

- Orderly and incremental synchronization

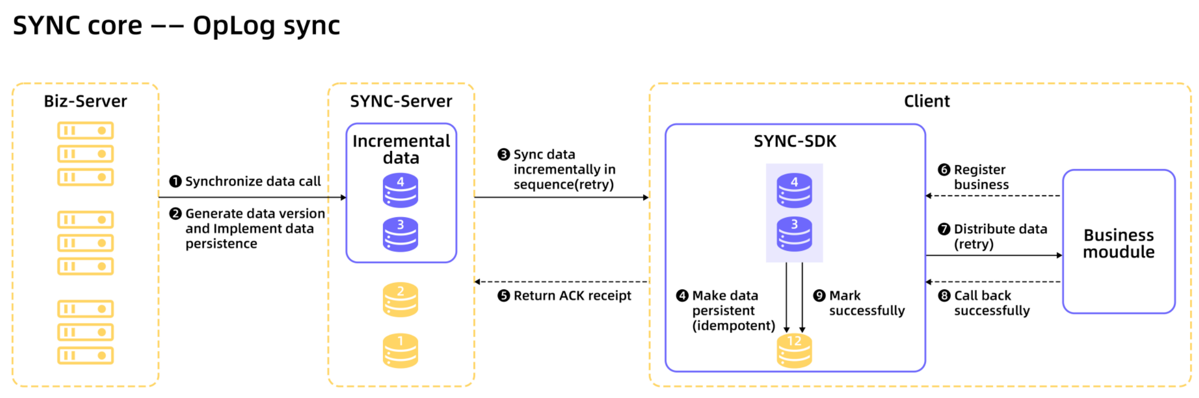
  MSS ensures that messages transmitted in the same channel arrive at the client in the same sequence as your server calls the MSS server, and all messages are synchronized to the client on an incremental basis.

- Highly real-time performance

  When the network connection of the client is good, MSS can ensure highly real-time push performance. The time taken for message synchronization almost equals the time taken for pure data transmission over the network (that is, messages can arrive within 1s).

## Basic principles

Similar to the binlog mechanism in MySQL, the basic data unit transmitted between the MSS server and the client SDK is oplog. To synchronize a piece of changed data to a specified user or device, your server needs to call the MSS API. Then the MSS server packages the data as an oplog and stores it in the database. When the client is online, the MSS server synchronizes the oplog to the client. Each oplog has a unique ID. Oplog IDs are unique and monotonically increments (based on the call sequence) among certain users and within a certain business scope. The MSS server synchronizes all oplogs to the client in ascending order of oplog ID. Both the MSS server and the client record the largest oplog ID received by the client, which is called the synchronization point (or understood as the data version number).



## Advantages

- Merged push

  When the client is successfully initialized, the server can push multiple pieces of business data at a time to reduce the number of requests.

- Incremental push

  Only incremental data is synchronized, reducing the transmission of redundant data and the network costs.

- Reduced requests

  Data synchronization is not requested when there is no incremental data, reducing redundant requests.

- Improved time efficiency

  When the server encounters data changes, the changed data is instantly synchronized to the client, without the need to wait for requests from the client.

- Improved experience

  Data is synchronized imperceptibly and is present before the client UI is rendered, reducing the waiting time of users.

## Applicable scenarios

MSS can be applied in business scenarios where data needs to be synchronized in real time to the client, such as transfer result synchronization, payment result synchronization, and message center. You can learn more about MSS capabilities through the following scenarios:

- In instant messaging Apps, MSS provides incremental and reliable message delivery capabilities to synchronize chat messages to specified users based on the message sending order of the sender.
- In Apps requiring dynamic configuration updates, MSS dynamically synchronizes configuration information to all devices. MSS synchronizes information including the app function switch, dynamic parameters, and dynamic configurations to the specified client in real time, or dynamically modifies the business parameters and configurations in batches when the App is running.
- For payment Apps, MSS provides a secure data channel for synchronizing transaction data online, ensuring that the Apps can receive the data in real time when they are online. In addition, MSS provides the data persistence capability. If data is synchronized when an App is offline, the App can receive the data when going online.

# 3.Terms

The terms in this glossary are sorted alphabetically.

## B

### Background

Background refers to the state where the client app is running but is not the active application. This occurs when the user is on the phone's home screen, using another app, or the phone screen is off.

### BizType

BizType is a unique identifier for a business scenario. The client MSS SDK uses the BizType to distribute pushed data to the corresponding business module.

### Business dimension

Business dimension refers to the scope of MSS. It can be either the user dimension or the device dimension. The user dimension pushes data based on a userId. The device dimension pushes data based on a utdId.

## F

### Foreground

The client app is running.

## I

### Idempotence

Idempotence ensures that duplicate data is discarded based on the `thirdMsgId` field in `SyncOrder`. The combination of `bizType`, `linkToken`, and `thirdMsgId` must be unique. This allows an operation to succeed only once. If a duplicate operation is attempted, the API call returns a success message with the result code DUPLICATED_BIZ_ID.

## M

### Multi-device synchronization

With multi-device synchronization, data pushed based on the user dimension is synchronized across all of a user's devices. A user who switches devices will still receive data that was previously delivered to another device. If the user uninstalls and reinstalls the client, the data is pushed again after the user comes online.

### MSS data

This refers to data that is pushed by the MSS server.

### MSS push

MSS push is the process of actively sending data from the server-side to a client. If the client is online, the push is triggered immediately. If the client is offline, the data is pushed after the client comes online.

## O

### Online

A client app is considered online when it has a network connection and can maintain a stable, long-lived TCP connection.

Most Android phones allow an app to stay online while in the background. On iOS devices, an app can stay online in the background for three minutes due to operating system limitations.

## P

### Persistence

Persistence is the mechanism that converts program data between persistent and transient states. In the MSS service, this mechanism enables two data behaviors: persistent and non-persistent.

- Persistent data: When a user or device is offline, the data is persisted to the database. After the user or device comes online, the MSS SDK triggers synchronization.
- Non-persistent data: When a user or device is online, the data is pushed immediately. If they are offline, the data is discarded. The user cannot receive this data after coming online again.

### Push type

There are two push types: targeted push and global push.

- Targeted push: Pushes data to a specific userId or utdId.
- Global push: Pushes data to all online users or devices. Global push uses multi-device synchronization.

## S

### Single-device push

Single-device push sends a message only once to the user's most recently online device when data is pushed based on the user dimension. The system does not resend the data if the user uninstalls and reinstalls the client or comes online on a different device.

### SYNC

SYNC is the MSS service that synchronizes data from the server-side to the client app.

## T

### Threshold

A threshold is the maximum amount of backlogged data allowed on the server. A data backlog can occur if a user or device is offline for an extended period while the MSS server continues to generate new data. When the backlog exceeds the threshold, older data is discarded, and only the most recent data is retained.

# 4.Client-side development

## 4.1. Integrate with Android

This topic describes how to integrate the MSS component with an Android client.

**Note**

As of June 28, 2020, mPaaS no longer maintains the 10.1.32 baseline. Use the 10.1.68 or 10.1.60 baseline. To upgrade your baseline version, see mPaaS 10.1.68 Upgrade Guide or mPaaS 10.1.60 Upgrade Guide.

The MSS component supports two integration methods: **native AAR-based integration** and **component-based integration**.

The integration procedure consists of two steps:

1. Add the SDK
2. Use the SDK

### Prerequisites

You have integrated your project with mPaaS.

- For native AAR-based integration, add mPaaS to your project.
- For component-based integration, complete the component-based integration process.

### Add the SDK

### Native AAR-based integration

Use **Component Management (AAR)** to install the **SYNC** component in your project. For more information, see AAR component management.

### Component-based integration

In the Portal and Bundle projects, use **Component Management** to install the **SYNC** component.

For more information, see Manage component dependencies > Add or remove component dependencies.

### Use the SDK

In baselines 10.1.32 and later, the `MPSync` class in the mPaaS middleware layer encapsulates all APIs of the MSS component. You can use the `MPSync` object to access all MSS features.

The following table provides a quick overview of the MSS APIs. For more information about the APIs, see Android API reference.

| API | Description |
|---|---|
| setup(Application application) | Initializes the basic services that MSS depends on. This method must be called before the `initialize` method. This method is available only in baselines 10.1.60 and later. |

| | |
|---|---|
| initialize(Context context) | Initializes the API and the MSS service. |
| appToForeground() | Notifies the client SDK that the app is started to establish a network connection with the server. Call this method each time the app is brought to the foreground. |
| appToBackground() | Notifies the client SDK that the app is sent to the background to disconnect from the server. Call this method each time the app is sent to the background. |
| updateUserInfo(String sessionId) | Call this method when the logon information such as userId or sessionId changes. This method must be called at least once. |
| clearUserInfo() | Logs out the user. |
| registerBiz(String bizType, ISyncCallback syncCallback) | Registers a `callback` to receive business data. After the pushed data is received, the client SDK calls the `syncCallback` implementation class. |
| unregisterBiz(String bizType) | Unregisters the specified synchronization configuration. After the configuration is unregistered, the client SDK no longer calls the `syncCallback` implementation class when pushed data is received. |
| reportMsgReceived(SyncMessage syncMessag) | After data is received in the `syncCallback` implementation class, call this API to notify the MSS server that the data is successfully received. If the server does not receive a `reportMsgReceived` call, it retries to deliver the data. After six retries, the data is permanently deleted. |
| isConnected() | Checks whether the MSS service is running as expected. |

## Code sample

This sample is developed using the SDK of baseline 10.1.32. In the sample application, a button is configured to obtain the device ID when tapped. The console then uses the device ID to push synchronization data to the device. The synchronization identity is `bizType`.

**Note**

This sample only demonstrates how to call the MSS APIs and is not a best practice for MSS. Download the best practice code for the MSS component from the Get code sample page.

```
    public void button1Clicked(View view)
    {
        // Use the getUtdid method to obtain the device ID.
        String utdid =UTDevice.getUtdid(MainActivity.this);
        // Print the MSS data in Logcat.
        Log.e("=========",utdid);
        // Initialize the API and the MSS service.
        MPSync.initialize(MainActivity.this);
        // Register a callback to receive business data. After the pushed data is recei
ved, the syncCallback is called.
        MPSync.registerBiz("bizType",new SyncCallBackImpl());
        // Establish a network connection with the server.
        MPSync.appToForeground();

    }

    public class SyncCallBackImpl implements ISyncCallback
    {
        @Override
        public void onReceiveMessage(SyncMessage syncMessage) {
            // Print the MSS data in Logcat.
            Log.e("=========",syncMessage.msgData);
            // Notify the MSS server that the data is successfully received.
            MPSync.reportMsgReceived(syncMessage);
        }
    }
```

### TLS support

To enable TLS connections on ports other than 443, add the metadata `mss_use_tls` to the `manifest` file and set its value to `YES`.

```
<meta-data
    android:name="mss_use_tls"
    android:value="YES"
    />
```

**Note**

Connections on port 443 use TLS by default.

### What to do next

Integrate with the server-side

## 4.2. Integrate with iOS

### 4.2.1. Add SDK

This topic describes how to quickly integrate the Mobile Sync Service (MSS) component into an iOS client.

The Mobile Sync Service supports three connection types: integration based on the mPaaS framework, integration into an existing project using the mPaaS plug-in, and integration into an existing project using CocoaPods. For more information, see Introduction to connection types. Select a connection type as needed.

## Prerequisites

You have integrated your project with mPaaS. For more information, see the following topics:

- Integration into an existing project using CocoaPods

## Add the SDK

Select an approach to add the SDK based on your connection type.

### Use the mPaaS Xcode Extension

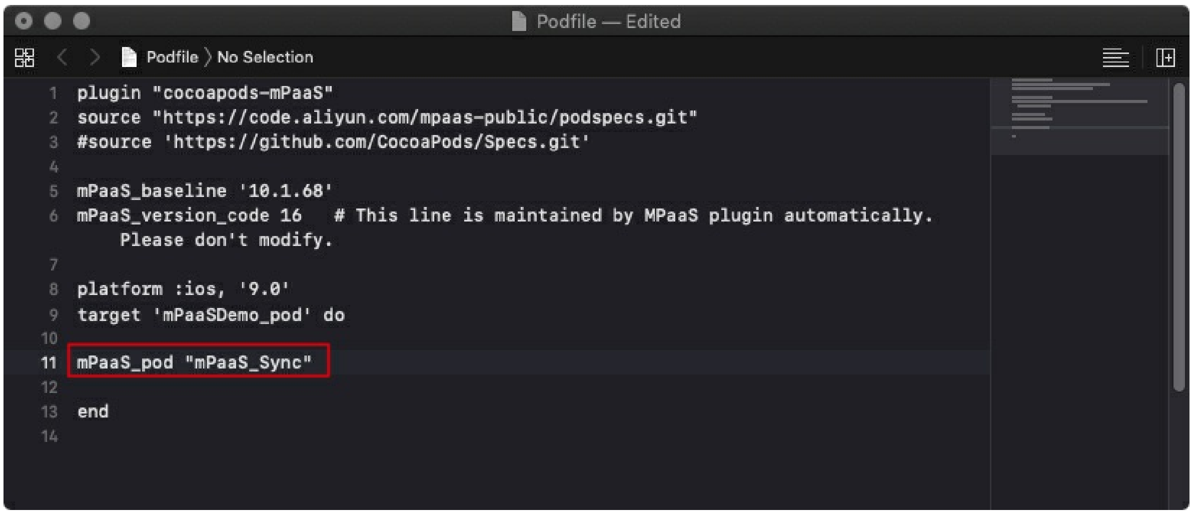
This approach applies if you are integrating the SDK **based on the mPaaS framework** or **into an existing project using the mPaaS plug-in**.

1. In the Xcode menu bar, choose **Editor** > **mPaaS** > **Edit Project** to open the project editing page.
2. Select **Mobile Sync Service**, save the configuration, and then click **Start Editing** to add the component.

### Use the cocoapods-mPaaS plug-in

This approach applies if you are integrating the SDK **into an existing project using CocoaPods**.

1. In the Podfile, add the `mPaaS_pod "mPaaS_Sync"` dependency for the MSS component.



```
plugin "cocoapods-mPaaS"
source "https://code.aliyun.com/mpaas-public/podspecs.git"
#source 'https://github.com/CocoaPods/Specs.git'

mPaaS_baseline '10.1.68'
mPaaS_version_code 16   # This line is maintained by MPaaS plugin automatically.
    Please don't modify.

platform :ios, '9.0'
target 'mPaaSDemo_pod' do

mPaaS_pod "mPaaS_Sync"

end

```

2. Run `pod install` to complete the integration.

## 4.2.2. Use SDK

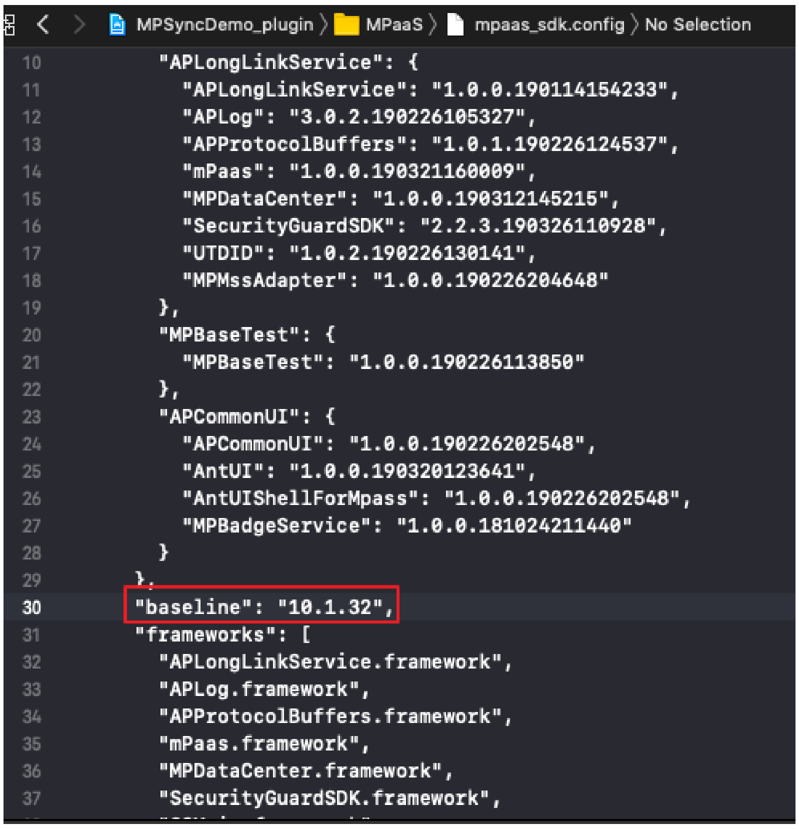
After adding the MSS SDK, project configuration is required before using the SDK.

> **Note**
>
> As of June 28, 2020, mPaaS no longer maintains the 10.1.32 baseline. Use the 10.1.68 or 10.1.60 baseline series instead. To upgrade your baseline, see the mPaaS 10.1.68 Upgrade Guide or mPaaS 10.1.60 Upgrade Guide.

## Prerequisites

Ensure that you use SDK version 10.1.32 or later.

> **Note**
>
> You can find the current SDK version in the `mpaas_sdk.config` file.



MPSyncDemo_plugin 〉 MPaaS 〉 mpaas_sdk.config 〉 No Selection

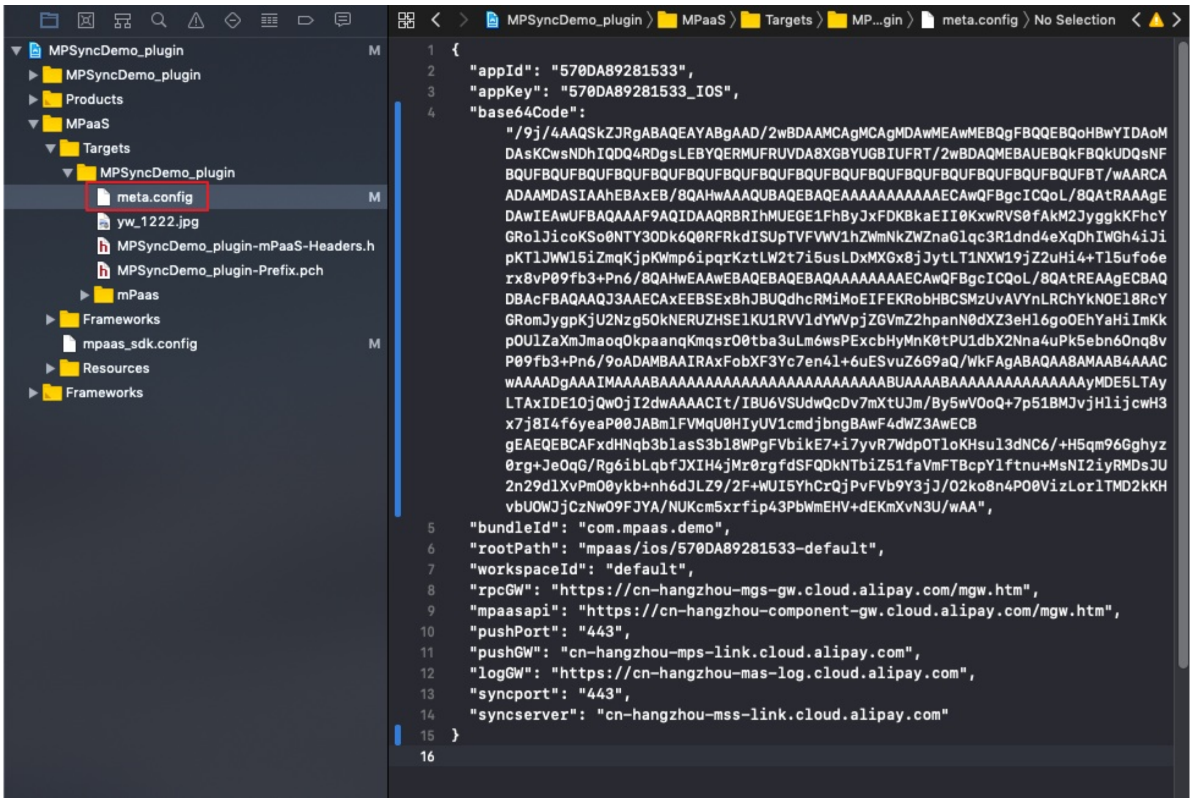
```
        "APLongLinkService": {
          "APLongLinkService": "1.0.0.190114154233",
          "APLog": "3.0.2.190226105327",
          "APProtocolBuffers": "1.0.1.190226124537",
          "mPaas": "1.0.0.190321160009",
          "MPDataCenter": "1.0.0.190312145215",
          "SecurityGuardSDK": "2.2.3.190326110928",
          "UTDID": "1.0.2.190226130141",
          "MPMssAdapter": "1.0.0.190226204648"
        },
        "MPBaseTest": {
          "MPBaseTest": "1.0.0.190226113850"
        },
        "APCommonUI": {
          "APCommonUI": "1.0.0.190226202548",
          "AntUI": "1.0.0.190320123641",
          "AntUIShellForMpass": "1.0.0.190226202548",
          "MPBadgeService": "1.0.0.181024211440"
        }
      },
      "baseline": "10.1.32",
      "frameworks": [
        "APLongLinkService.framework",
        "APLog.framework",
        "APProtocolBuffers.framework",
        "mPaas.framework",
        "MPDataCenter.framework",
        "SecurityGuardSDK.framework",
```

## Configure the project

Ensure that the `meta.config` file is in your project. This file contains configuration information, such as the Sync service endpoint and port.

- If you use the latest plug-in to add the Sync SDK, this file is generated automatically.
- If the `meta.config` file does not exist in your project,

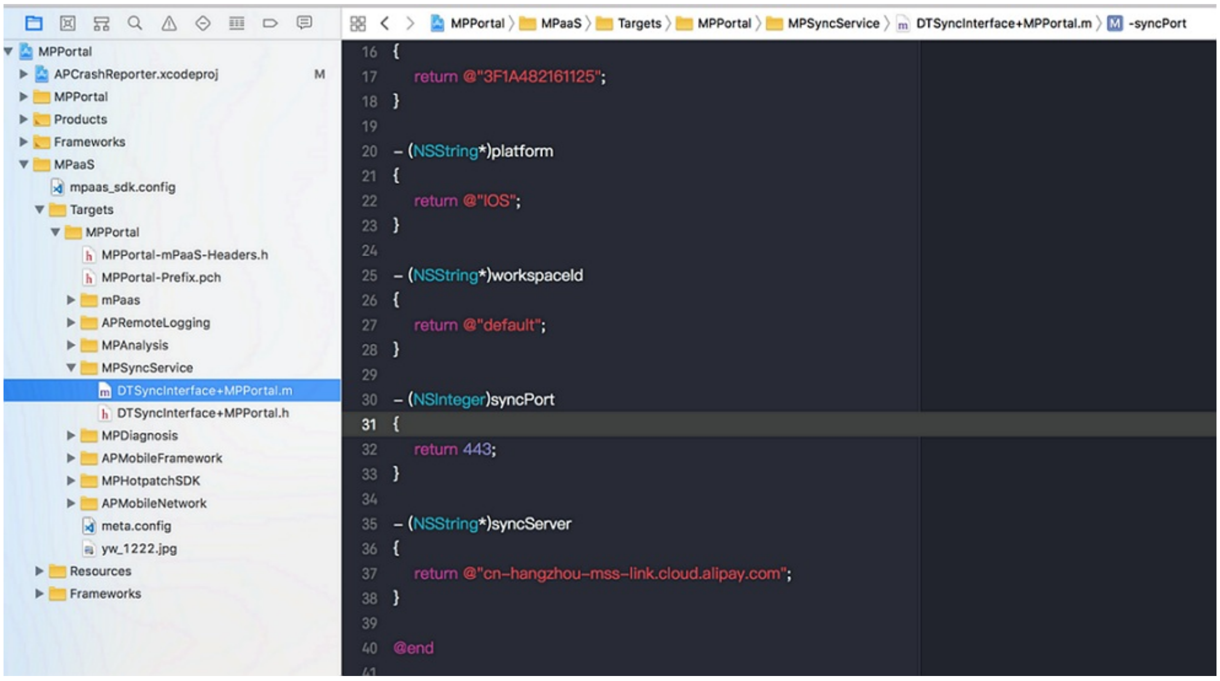
  Go to the **Console** > **Code Management** > **Code Configuration** page to download the `.config` configuration file. Then, rename the file to `meta.config` and add it to your project.



```
{
  "appId": "570DA89281533",
  "appKey": "570DA89281533_IOS",
  "base64Code":
      "/9j/4AAQSkZJRgABAQEAYABgAAD/2wBDAAMCAgMCAgMDAwMEAwMEBQgFBQQEBQoHBwYIDAoM
      DAsKCwsNDhIQDQ4RDgsLEBYQERMUFRUVDA8XGBYUGBIUFRT/2wBDAQMEBAUEBQkFBQkUDQsNF
      BQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBT/wAARCA
      ADAAMDASIAAhEBAxEB/8QAHwAAAQUBAQEBAQEAAAAAAAAAAAECAwQFBgcICQoL/8QAtRAAAgE
      DAwIEAwUFBAQAAAF9AQIDAAQRBRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0fAkM2JyggkKFhcY
      GRolJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqDhIWGh4iJi
      pKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uHi4+Tl5ufo6e
      rx8vP09fb3+Pn6/8QAHwEAAwEBAQEBAQEBAQAAAAAAAAECAwQFBgcICQoL/8QAtREAAgECBAQ
      DBAcFBAQAAQJ3AAECAxEEBSExBhJBUQdhcRMiMoEIFEKRobHBCSMzUvAVYnLRChYkNOEl8RcY
      GRomJygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6goOEhYaHiImKk
      pOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4uPk5ebn6Onq8v
      P09fb3+Pn6/9oADAMBAAIRAxFobXF3Yc7en4l+6uESvuZ6G9aQ/WkFAgABAQAA8AMAAB4AAAC
      wAAAADgAAAIMAAAABAAAAAAAAAAAAAAAAAAAAAAAAABUAAAABAAAAAAAAAAAAAAyMDE5LTAy
      LTAxIDE1OjQwOjI2dwAAAACIt/IBU6VSUdwQcDv7mXtUJm/By5wVOoQ+7p51BMJvjHlijcwH3
      x7j8I4f6yeaP00JABmlFVMqU0HIyUV1cmdjbngBAwF4dWZ3AwECB
      gEAEQEBCAFxdHNqb3blasS3bl8WPgFVbikE7+i7yvR7WdpOTloKHsul3dNC6/+H5qm96Gghyz
      0rg+JeOqG/Rg6ibLqbfJXIH4jMr0rgfdSFQDkNTbiZ51faVmFTBcpYlftnu+MsNI2iyRMDsJU
      2n29dlXvPmO0ykb+nh6dJLZ9/2F+WUI5YhCrQjPvFVb9Y3jJ/O2ko8n4PO0VizLorlTMD2kKH
      vbUOWJjCzNwO9FJYA/NUKcm5xrfip43PbWmEHV+dEKmXvN3U/wAA",
  "bundleId": "com.mpaas.demo",
  "rootPath": "mpaas/ios/570DA89281533-default",
  "workspaceId": "default",
  "rpcGW": "https://cn-hangzhou-mgs-gw.cloud.alipay.com/mgw.htm",
  "mpaasapi": "https://cn-hangzhou-component-gw.cloud.alipay.com/mgw.htm",
  "pushPort": "443",
  "pushGW": "cn-hangzhou-mps-link.cloud.alipay.com",
  "logGW": "https://cn-hangzhou-mas-log.cloud.alipay.com",
  "syncport": "443",
  "syncserver": "cn-hangzhou-mss-link.cloud.alipay.com"
}
```

## Notes for upgrading from an earlier version

Versions later than 10.1.32 do not require the `Category` file for the `DTSyncInterface` class. Instead, the middle layer reads the configuration from the `meta.config` file. After you upgrade, check your project for legacy configurations and remove them.



```
{
    return @"3F1A482161125";
}

- (NSString*)platform
{
    return @"IOS";
}

- (NSString*)workspaceId
{
    return @"default";
}

- (NSInteger)syncPort
{
    return 443;
}

- (NSString*)syncServer
{
    return @"cn-hangzhou-mss-link.cloud.alipay.com";
}

@end
```

## Code example

To listen for synchronization events, create a class. A RAM-resident service is best for continuously listening for Sync messages. In the following example, a `MySyncService` class is created to listen for synchronization events.

Before listening for synchronization events, define a business identity name. This name corresponds to the synchronization configuration identity in the data synchronization console. This parameter identifies your client to the service. In the following example, the business identity name is `SYNC-TRADE-DATA` .

```
#import <MPMssAdapter/MPSyncInterface.h>
#define SYNC_BIZ_NAME @"SYNC-TRADE-DATA"

@implementation MySyncService
+ (instancetype)sharedInstance
{
    static MySyncService *bizService;

    static dispatch_once_t llSOnceToken;

    dispatch_once(&llSOnceToken, ^{

        bizService = [[self alloc] init];
    });
    return bizService;
}

-(instancetype)init
{
    self = [super init];
    if (self) {
        [MPSyncInterface initSync];
        BOOL registerSingleDeviceSync = [MPSyncInterface
registerSyncBizWithName:SYNC_BIZ_NAME syncObserver:self
selector:@selector(revSyncBizNotification:)];
        [MPSyncInterface bindUserWithSessionId:@"SESSION_DEMO"]; // The User
corresponds to the userId required when sending commands from the console, and must mat
ch the value in the MPaaSInterface userId function. The sessionId is the client's autho
rization token. Both userId and sessionId are returned on user logon. If either value c
hanges, call this function again to re-establish the persistent connection.
    }
    return self;
}

-(void)revSyncBizNotification:(NSNotification*)notify
{
    NSDictionary *userInfo = notify.userInfo;
    dispatch_async(dispatch_get_main_queue(), ^{
        // Process the business data
        [MySyncService handleSyncData:userInfo];
        // Callback to the Sync SDK to indicate that the business data is processed.
        [MPSyncInterface responseMessageNotify:userInfo];
    });
}

+(void)handleSyncData:(NSDictionary *)userInfo
```

```
{
    NSString * stringOp = userInfo[@"op"];
    NSArray *op = [NSJSONSerialization JSONObjectWithData:[stringOp
dataUsingEncoding:NSUTF8StringEncoding] options:NSJSONReadingMutableContainers error:ni
l];
    if([op isKindOfClass:[NSArray class]]){
        [op enumerateObjectsUsingBlock:^(NSDictionary * item, NSUInteger idx, BOOL
*stop) {
            if([item isKindOfClass:[NSDictionary class]]){
                NSString * plString = item[@"pl"];// Business data payload
                if(item[@"isB"]){
                    NSData *dataPl = [[NSData alloc]
initWithBase64EncodedString:plString options:kNilOptions];
                    NSString *pl = [[NSString alloc] initWithData:dataPl encoding:NSUTF8
StringEncoding];
                    NSLog(@"biz payload data:%@,string:%@",dataPl,pl);
                }else{
                     NSLog(@"biz payload:%@",plString);
                }
            }
        }];
    }
}

-(void)dealloc
{
    BOOL unRegisterSingleDeviceSync = [MPSyncInterface
unRegisterSyncBizWithName:SYNC_BIZ_NAME syncObserver:[MySyncService sharedInstance]];
    [MPSyncInterface removeSyncNotificationObserver:self];
}
@end
```

# 5.Server-side Integration

## 5.1. Server-side integration guide

This topic describes the server-side development process for integrating a tenant's business system with Mobile Sync Service (MSS).

The process involves the following two steps:

1. Integrate the Java SDK and write call code: Integrate the Java SDK and write the call code. You can use either the single data sync API or the global data sync API.
2. Verify user consistency: This verification is typically used for business scenarios that require a high level of security for synchronized data.

### Prerequisites

- You have activated the mPaaS product and obtained the AccessKey and AccessSecret from your tenant administrator.
- You have created an app and obtained its appId and workspaceId.
- You have a server-side application.
- You have configured Maven.

## 5.2. Connect using the Java SDK

This topic describes how to integrate the MSS on the server-side using the Java SDK.

### Import the JAR package

After you configure Maven, add the following dependencies to the main `pom.xml` file:

**Note**

For users outside the finance zone, the latest version of the Message Push V2.0 SDK is 5.0.2. For users in the finance zone, the latest version of the Message Push V2.0 SDK is 2.1.11.

```
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>mpaas20201028</artifactId>
    <version>5.0.1</version>
</dependency>
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>tea-openapi</artifactId>
    <version>0.3.6</version>
</dependency>
```

### Configure environment variables

**Configure the MPAAS_AK_ENV and MPAAS_SK_ENV environment variables.**

- For Linux and macOS, run the following commands:

```
export MPAAS_AK_ENV=<access_key_id>
export MPAAS_SK_ENV=<access_key_secret>
```

> **Note**
>
> Replace `access_key_id` with your AccessKey ID and `access_key_secret` with your AccessKey secret.

- Windows system configuration
  1. Create the **MPAAS_AK_ENV** and **MPAAS_SK_ENV** environment variables. Set their values to your AccessKey ID and AccessKey secret.
  2. Restart Windows.

## API reference

## Single data synchronization API

The single data synchronization API syncs data to a specific user or device.

## Parameter descriptions

The parameters are as follows:

| Name | Type | Required | Example | Description |
|---|---|---|---|---|
| appId | String | Yes | ONEX570DA892117 | The App ID obtained from the mPaaS console. |
| workspaceId | String | Yes | PROD | The Workspace ID obtained from the mPaaS console. |
| bizType | String | Yes | UCHAT | The synchronization identity configured in the mPaaS console. For more information, see Console overview. |
| linkToken | String | Yes | | The ID of the push target. If you push data based on users, enter the user ID. If the synchronization is configured to push data based on devices, enter the device ID. |
| payload | String | Yes | testtestatapalayd | The actual business message body. The format is customizable. The length cannot exceed 4,096 characters. |

| | | | | |
|---|---|---|---|---|
| thirdMsgId | String | Yes | 1760339273 | The ID of a data synchronization request. The ID must be unique within the same synchronization identity. Requests with duplicate IDs are ignored. The ID must be less than 100 bytes. |
| osType | String | No | iOS/Android | Filters pushes by mobile platform. By default, this parameter is not passed, and no filtering is performed. Data is pushed to both iOS and Android platforms. |
| appMinVersion | String | No | 0.0.0.0 | Filters pushes by client version. Data is pushed only to clients with a version number greater than or equal to this value. |
| appMaxVersion | String | No | 100.100.100.100 | Filters pushes by client version. Data is pushed only to clients with a version number less than or equal to this value. |
| validTimeStart | String | No | 1584448493913 | Data is pushed only when the current time is greater than or equal to `validTimeStart`. |
| validTimeEnd | String | No | 1584452093913 | Data is pushed only when the current time is less than or equal to `validTimeEnd`. |

## Result codes for single data synchronization

| Result code | Description | Solution |
|---|---|---|
| SUCCESS | Success | Success |
| ARGS_IS_NULL | A required parameter is empty. | Check that all required parameters are provided. |
| PAYLOAD_LONG | The message payload is too long. | Check that the length of the payload parameter does not exceed the limit. |
| THIRD_MSG_ID_LONG | The third-party business ID is too long. | Check that the length of the third-party business ID does not exceed the limit. |

| | | |
|---|---|---|
| BIZ_NOT_ONLINE | The synchronization identity for the business scenario is not published. | Go to the **mPaaS console** > **Data Synchronization**. Check that the synchronization identity that corresponds to `bizType` is configured and published. |
| THIRD_MSG_ID_IS_NULL | The third-party business ID is empty. | Check that the third-party business ID is not empty. |
| SYSTEM_ERROR | A system error occurred. | Contact technical support to identify the cause of the error. |
| INVALID_TENANT_ID | The tenant ID is invalid. | Check that the appId is correct and that you have the permission to use it. |

## Code example

```
import com.alibaba.fastjson.JSON;
import com.aliyun.mpaas20201028.Client;
import com.aliyun.mpaas20201028.models.CreateOpenSingleDataRequest;
import com.aliyun.mpaas20201028.models.CreateOpenSingleDataResponse;
import com.aliyun.teaopenapi.models.Config;

public static void main(String[] args) throws Exception {
    // An Alibaba Cloud account AccessKey has full access to all APIs. We recommend tha
t you use a Resource Access Management (RAM) user for API calls and routine O&M.
    // To prevent AccessKey leaks and protect your resources, do not store your
AccessKey ID and AccessKey secret in your project code.
    // This example shows how to store the AccessKey ID and AccessKey secret in environ
ment variables. You can also store them in a configuration file as needed.
    Config config = new Config();
    // Required. Your AccessKey ID.
    config.setAccessKeyId(System.getenv("MPAAS_AK_ENV"));
    // Required. Your AccessKey secret.
    config.setAccessKeySecret(System.getenv("MPAAS_SK_ENV"));
    // The REGION_ID and Endpoint of mPaaS. This example uses a non-finance region in H
angzhou.
    config.setRegionId("cn-hangzhou");
    config.setEndpoint("mpaas.cn-hangzhou.aliyuncs.com");
    Client client = new Client(config);

    CreateOpenSingleDataRequest singleRequest = new CreateOpenSingleDataRequest();
    //*************Required properties*************/

    // The App ID obtained from the mPaaS console.
    singleRequest.setAppId("xxxxxxx");
    // The Workspace ID obtained from the mPaaS console.
    singleRequest.setWorkspaceId("xxxxxxxx");
    // The synchronization identity configured in Mobile Sync in the mPaaS console.
    singleRequest.setBizType("TEST-SYNC");
    // The ID of the user or device (UTDID) to push to.
    singleRequest.setLinkToken("testUserId");
    // The actual business message body. The custom length cannot exceed 4,096 characte
rs.
    singleRequest.setPayload("testPayload");
    // The business ID. It must be unique and cannot exceed 100 characters in length.
    singleRequest.setThirdMsgId("test_third_msg_id_" + System.currentTimeMillis());

    //************Optional properties*************/

    // The OS of the target device. Valid values: iOS and Android. If left empty, the O
S is not restricted.
    singleRequest.setOsType("IOS");
    // The minimum client version number, such as 8.6.0.0.9999. If left empty, the mini
mum version is not restricted.
    singleRequest.setAppMinVersion("0.0.0.0");
    // The maximum client version number, such as 9.0.0.0.9999. If left empty, the maxi
mum version is not restricted.
    singleRequest.setAppMaxVersion("100.100.100.100");
    // The start of the validity period. If left empty, the start time is not restricte
d.
    singleRequest.setValidTimeStart(System.currentTimeMillis());
    // The end of the validity period. If left empty, the end time is not restricted. T
he maximum validity period is 30 days.
    singleRequest.setValidTimeEnd(System.currentTimeMillis() + (1000 * 3600));

    CreateOpenSingleDataResponse openSingleData =
client.createOpenSingleData(singleRequest);
    System.out.println("response==>"+JSON.toJSONString(openSingleData));
}
```

**Important**

Make sure your AccessKey has the `AliyunMPAASFullAccess` permission. For more information, see Application-level access control for RAM accounts.

## Global data synchronization API

Global data synchronization synchronizes data to all devices.

## Parameter descriptions

The following are the business parameters:

| Name | Type | Required | Example | Description |
|---|---|---|---|---|
| appId | String | Yes | ONEX570DA892117 | The App ID obtained from the mPaaS console. |
| workspaceId | String | Yes | PROD | The Workspace ID obtained from the mPaaS console. |

| | | | | |
|---|---|---|---|---|
| bizType | String | Yes | UCHAT | The synchronization identity configured in the mPaaS console. For more information, see Console overview. |
| payload | String | Yes | testtestatapalayd | The actual business message body. The format is customizable. The length cannot exceed 4,096 characters. |
| thirdMsgId | String | Yes | 1760339273 | The ID of a data synchronization request. The ID must be unique within the same synchronization identity. Requests with duplicate IDs are ignored. The ID must be less than 100 bytes. |
| osType | String | No | IOS/ANDROID | Filters pushes by mobile platform. By default, this parameter is not passed, and no filtering is performed. Data is pushed to iOS and Android platforms. |
| appMinVersion | String | No | 0.0.0.0 | Filters pushes by client version. Data is pushed only to clients with a version number greater than or equal to this value. |
| appMaxVersion | String | No | 100.100.100.100 | Filters pushes by client version. Data is pushed only to clients with a version number less than or equal to this value. |
| validTimeStart | String | No | 1584448493913 | Data is pushed only when the current time is greater than or equal to \`validTimeStart\`. |
| validTimeEnd | String | No | 1584452093913 | Data is pushed only when the current time is less than or equal to \`validTimeEnd\`. |
| maxUid | Long | No | 99 | The maximum UID for the synchronization range (the second and third to last digits of the user ID or device ID). If the digits are not letters, convert them to ASCII codes. |
| minUid | Long | No | 00 | The minimum UID for the synchronization range (the second and third to last digits of the user ID or device ID). If the digits are not letters, convert them to ASCII codes. |

| | | | | |
|---|---|---|---|---|
| uids | String | No | 01,02,99 | This parameter has a higher priority than `maxUid` and `minUid`. Specifies discrete user ID segments (the second and third to last digits of the user ID or device ID). If the digits are not letters, convert them to ASCII codes. |

## Result codes for global data synchronization

| Result code | Description | Solution |
|---|---|---|
| SUCCESS | Success | Success |
| ARGS_IS_NULL | A required parameter is empty. | Check that all required parameters are provided. |
| PAYLOAD_LONG | The message payload is too long. | Check that the length of the payload parameter does not exceed the limit. |
| THIRD_MSG_ID_LONG | The third-party business ID is too long. | Check that the length of the third-party business ID does not exceed the limit. |
| BIZ_NOT_ONLINE | The synchronization identity for the business scenario is not published. | Go to the **mPaaS console** > **Data Synchronization**. Check that the synchronization identity that corresponds to `bizType` is configured and published. |
| THIRD_MSG_ID_IS_NULL | The third-party business ID is empty. | Check that the third-party business ID is not empty. |
| SYSTEM_ERROR | A system error occurred. | Contact technical support to identify the cause of the error. |
| NOT_SUPPORT_GLOBAL | The synchronization identity does not support global calls. | Go to the **mPaaS console** > **Data Synchronization**. Check if the synchronization identity that corresponds to the `BizType` is configured for pushing to a specific user or device. |
| INVALID_TENANT_ID | The tenant ID is invalid. | Check that the appId is correct and that you have the permission to use it. |

## Code example

```
import com.alibaba.fastjson.JSON;
import com.aliyun.mpaas20201028.Client;
import com.aliyun.mpaas20201028.models.CreateOpenGlobalDataRequest;
import com.aliyun.mpaas20201028.models.CreateOpenGlobalDataResponse;
import com.aliyun.teaopenapi.models.Config;

public static void main(String[] args) throws Exception {
    // An Alibaba Cloud account AccessKey has full access to all APIs. We recommend tha
t you use a Resource Access Management (RAM) user for API calls and routine O&M.
    // To prevent AccessKey leaks and protect your resources, do not store your
AccessKey ID and AccessKey secret in your project code.
    // This example shows how to store the AccessKey ID and AccessKey secret in environ
ment variables. You can also store them in a configuration file as needed.
    Config config = new Config();
    // Required. Your AccessKey ID.
    config.setAccessKeyId(System.getenv("MPAAS_AK_ENV"));
    // Required. Your AccessKey secret.
    config.setAccessKeySecret(System.getenv("MPAAS_SK_ENV"));
    // The REGION_ID and Endpoint of mPaaS. This example uses a non-finance region in H
angzhou.
    config.setRegionId("cn-hangzhou");
    config.setEndpoint("mpaas.cn-hangzhou.aliyuncs.com");
    Client client = new Client(config);

    CreateOpenGlobalDataRequest globalRequest = new CreateOpenGlobalDataRequest();
    //*************Required properties*************/

    // The App ID obtained from the mPaaS console.
    globalRequest.setAppId("BE9C457161429");
    // The Workspace ID obtained from the mPaaS console.
    globalRequest.setWorkspaceId("sit");
    // The synchronization identity configured in Mobile Sync in the mPaaS console.
    globalRequest.setBizType("test-global");
    // The actual business message body. The custom length cannot exceed 4,096 characte
rs.
    globalRequest.setPayload("testtestata");
    // The business ID. It must be unique and cannot exceed 100 characters in length.
    globalRequest.setThirdMsgId("test_third_msg_id_" + System.currentTimeMillis());

    //************Optional properties*************/

    // The OS of the target device. Valid values: iOS and Android. If left empty, the O
S is not restricted.
    globalRequest.setOsType("IOS");
    // The minimum client version number, such as 8.6.0.0.9999. If left empty, the mini
mum version is not restricted.
    globalRequest.setAppMinVersion("0.0.0.0");
    // The maximum client version number, such as 9.0.0.0.9999. If left empty, the maxi
mum version is not restricted.

    globalRequest.setAppMaxVersion("100.100.100.100");
    // The maximum UID.
    globalRequest.setMaxUid(Long.valueOf(99));
    // The minimum UID.
    globalRequest.setMinUid(Long.valueOf(1));
    // The grayscale UIDs to push to, from 00 to 99. This is a string array.
    globalRequest.setUids("01,02,99");
    globalRequest.setValidTimeStart(System.currentTimeMillis());
    globalRequest.setValidTimeEnd(System.currentTimeMillis() + (1000 * 3600));
    CreateOpenGlobalDataResponse openGlobalData =
client.createOpenGlobalData(globalRequest);
    System.out.println("response==>"+JSON.toJSONString(openGlobalData));
}
```

**Important**

Make sure your AccessKey has the `AliyunMPAASFullAccess` permission. For more information, see Application-level access control for RAM accounts.

## 5.3. User consistency validation

Some business scenarios require high security for data synchronization. You must ensure that the target user for a push is the same as the currently logged-on user and has not been spoofed. MSS provides a user consistency validation feature to meet this requirement. You can enable this feature as needed. The basic principles of this feature are as follows:

- When a client connects to the server, it reports the user ID (userId) and authorization token (sessionId). The user logon system returns both the userId and sessionId. If the userId or sessionId changes, the client must call the relevant interfaces to ensure the persistent connection is established correctly.
- The server can call a consistency validation interface that the tenant implements. The tenant uses this interface to control the consistency check. MSS records the result of this check.
- For synchronization configurations with high security requirements, tenants can enable consistency validation. Data is pushed only to user devices that pass this validation. For configurations without consistency validation enabled, the validation result is ignored.

### Configure the consistency validation interface

This section describes how to configure the `com.antcloud.session.validate` consistency validation interface.

**Important**

After you configure the consistency validation interface in the mPaaS console, disable the **signature validation** feature for the RPC interface. Otherwise, the consistency check logic for mobile synchronization will not work correctly.

### Procedure

Log on to the mPaaS console, select the target application, and go to the **Mobile Gateway Service** > **API Management** page. Add an API. For more information, see Mobile Gateway > API Management.

### Interface name

Set the operationType of the API to `com.antcloud.session.validate`. The request parameters are as follows:

| Name | Type and length specifications | Required | Example | Description |
|---|---|---|---|---|
| instanceId | String | Yes | instancedemo | A string in the format of \`workspaceId_appId\`. |
| userId | String | Yes | 20880939 | User ID. |
| sessionId | String | Yes | kkdddd | The authorization token from the client. |

## Response parameters

Implement the consistency check logic to return data in JSON format. The following is an example:

```
{
        "resultCode": "OK",
        "resultMsg": "Operation is done successfully",
        "success": true,
        "result": {
            "sid": "kkdddd",
            "valid":true/false
        }
}
```

The properties are described as follows:

| Name | Type | Example | Description |
|---|---|---|---|
| success | boolean | true/false | Indicates whether the service call is successful. Returns `true` for success and `false` for failure. If the call fails, check the `returnCode` for the reason. For more information, see Business result codes. |
| returnCode | String | ERROR | Result code. |
| resultMsg | String | SYSTEM-ERROR | Result message. |
| sid | String | kkdddd | Authorization token or sessionId. |
| valid | boolean | true/false | Validation result. |

## Business result codes

| Result | Result code | Description |
|---|---|---|
| true | OK | The operation is successful. |
| false | OPERATION_ERROR | An OPERATION error occurred. This applies only to the `com.antcloud.session.validate` interface. |

# 6.Console operations

## 6.1. Console overview

Mobile Sync Service(MSS) console lets you manage push configurations and push business data. Each push configuration defines a specific business scenario, and a data push executes that scenario.

In the MSS console, you can perform the following operations:

- Add a push configuration
- Push business data
- View configuration details
- Modify configuration
- Unpublish a configuration
- View statistics for configuration pushes
- Manage services
- View MSS operation records

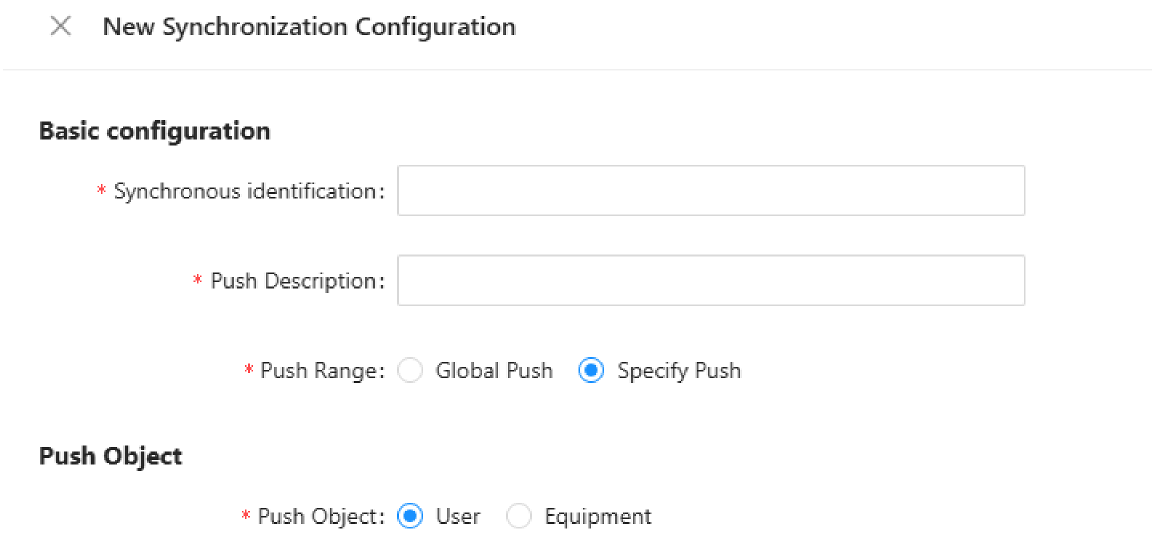
## 6.2. Add configuration

Each sync configuration represents a specific data push scenario that you can implement by pushing business data. Therefore, you must create a sync configuration before you push any business data.

In the **mPaaS** console, select the target application and follow these steps to add a configuration.

### Procedure

1. In the navigation pane on the left, click **Background connection** > **Mobile Sync Service** to go to the Data Synchronization page.
2. On the **Configuration Management** tab, click the **New Synchronization Configuration** button to go to the **New Synchronization Configuration** page.
3. On the page, set the parameters for each configuration item.


New Synchronization Configuration

Basic configuration

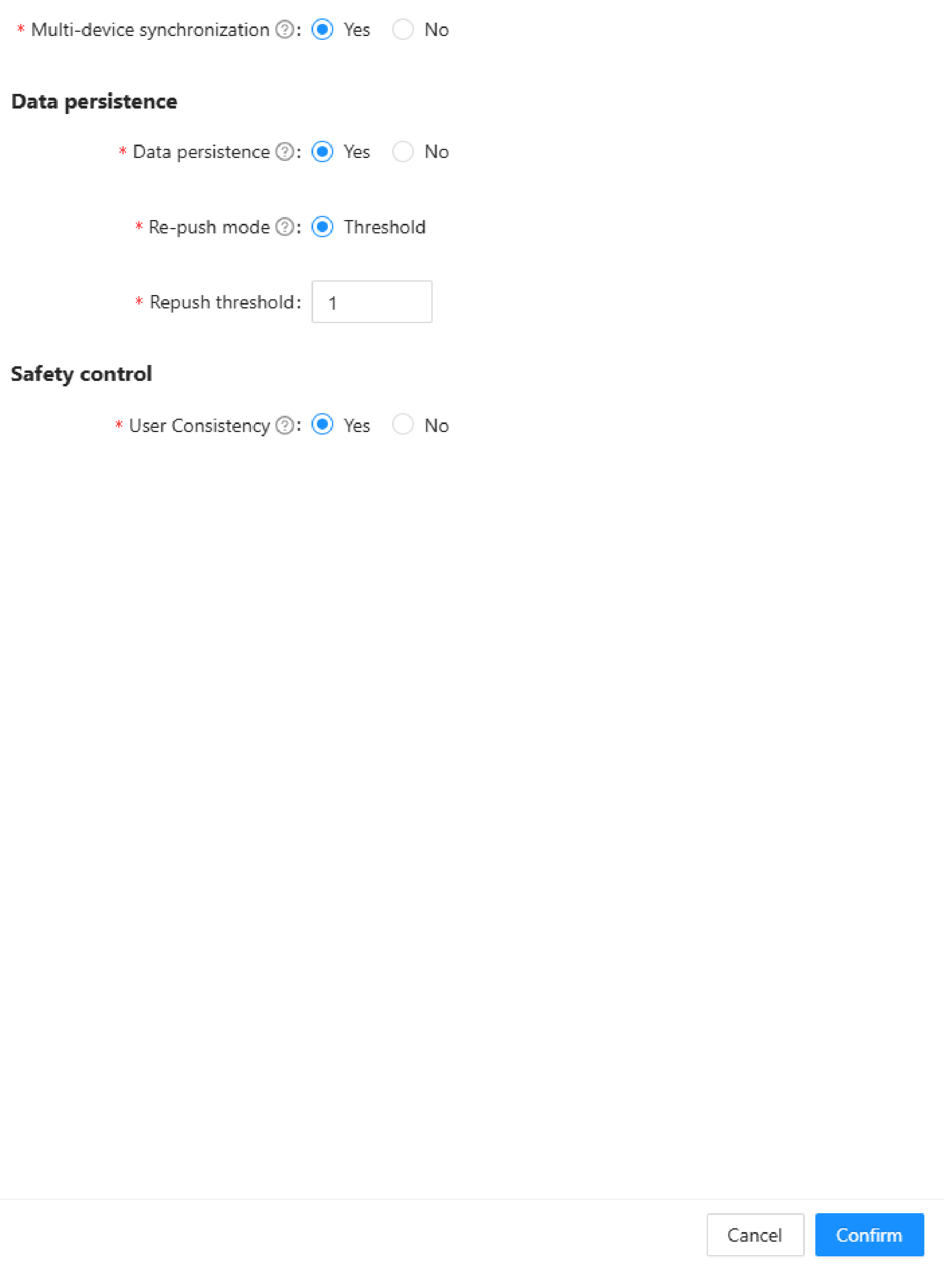
* Synchronous identification:

* Push Description:

* Push Range: Global Push [✓ Specify Push]

Push Object

* Push Object: [✓ User] Equipment



* Multi-device synchronization: [✓ Yes] No

**Data persistence**

* Data persistence: [✓ Yes] No

* Re-push mode: [✓ Threshold]

* Repush threshold: 1

**Safety control**

* User Consistency: [✓ Yes] No

The descriptions of each configuration item are shown in the table below.

| Configuration item | Description |
|---|---|

| | |
|---|---|
| Synchronous identification | Identifies a specific data push business scenario. Use uppercase English letters and the hyphen (-), such as DEVICE-LOCK. |
| Push Description | Enter remarks to describe the specific business scenario for this configuration. |
| Push Range | Specifies the range of users or devices that can receive data. Global push means all users or devices receive the data. Specified push means only a specific user or device receives the data. |
| Push Object | Indicates whether the data is pushed to the user or equipment. |
| Multi-device Synchronization | This option is required only when the push object is set to User. If you set this to Yes, data can be synchronized across multiple devices for a single user. This means that when a user switches devices, they will still receive data that was already delivered to their previous device. |
| Data Persistence | Indicates that pushed data is saved to the database. The default maximum retention period is 30 days. This prevents data loss. If a user is offline when data is pushed, they receive the data the next time they are online. |
| Re-push Mode | Specifies the policy for handling data that accumulates on the server-side. This setting is effective only when Data Persistence is set to Yes. Threshold push means that only the most recent accumulated data, up to the number specified by the threshold, is pushed to the client. |
| Repush Threshold | Effective only when Data Persistence is set to Yes and Re-push Mode is set to Threshold. |
| User Consistency | Effective only when the Push Object is set to User. When set to Yes, MSS validates user consistency when pushing data. The data is pushed only if the consistency requirements are met. Otherwise, the data is not pushed for this attempt. For more information, see User consistency validation. |

4. After you set the parameters, click **OK** to create the sync configuration. The new configuration is **Published** by default. After the configuration is published, you can push data by making API calls or by performing operations in the console.

## 6.3. Send business data

Go to the **mPaaS** console and select your target application. Then, follow these steps to add a data synchronization task to send business data.

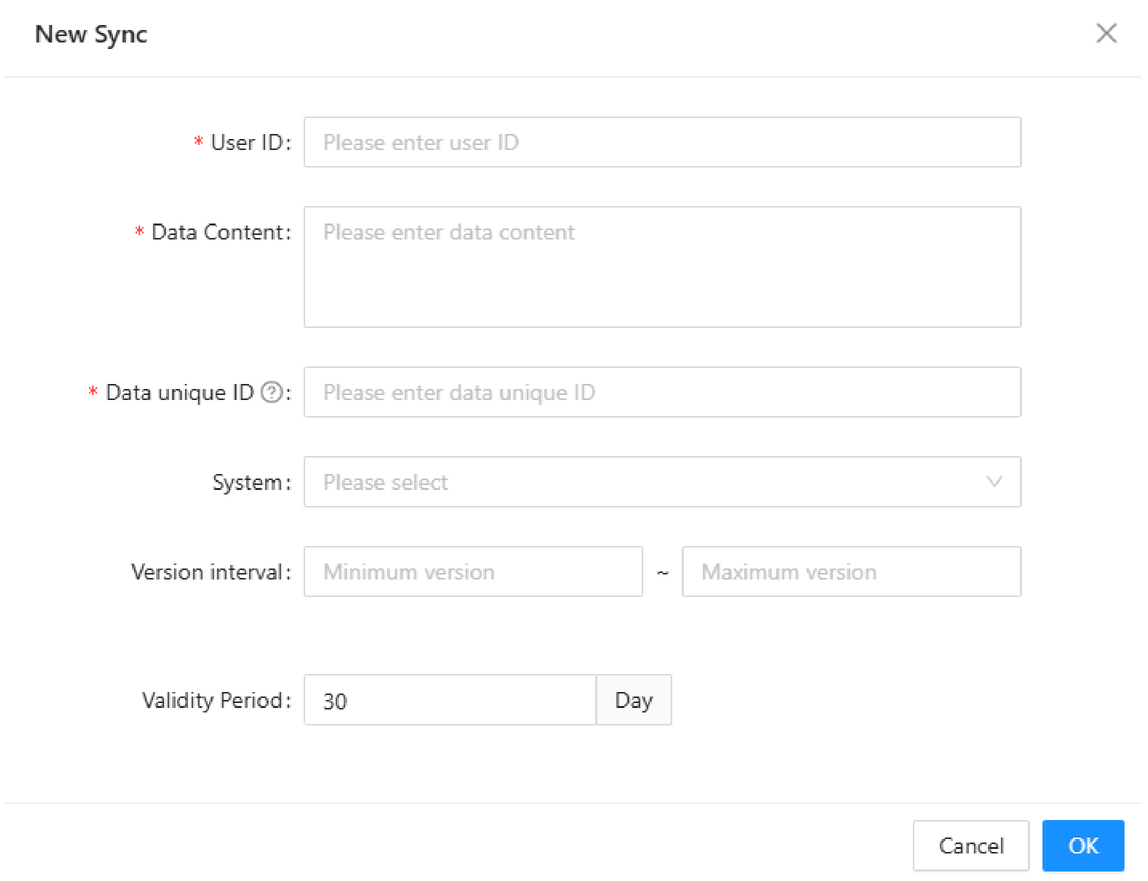
### Prerequisites

A push configuration has been created and published in the console.

### Procedure

1. In the navigation pane on the left, click **Background connection** > **Mobile Sync Service**
2. On the **Configuration Management** tab, find the target configuration and click **Operation**. The **New Sync** dialog box appears.

3. In the dialog box, enter the information for each configuration item and click **OK** to send the business data.


New Sync

* User ID: Please enter user ID

* Data Content: Please enter data content

* Data unique ID: Please enter data unique ID

System: Please select

Version interval: Minimum version ~ Maximum version

Validity Period: 30 Day

The following table describes the configuration items.

| Configuration item | Description |
|---|---|
| User ID/Device ID | Enter this information only for services that target specific users or device types. |
| Data content | The text content of the data, in string format. |
| Unique data ID | Required only for services that use data persistence. This ID uniquely identifies the data content. If two pushes have the same unique data ID, the second push is ignored. |
| System | The operating system of the client that will receive the data. By default, Android and iOS are selected. |
| Version interval | The application version interval for the clients that will receive the data. This parameter is optional. |

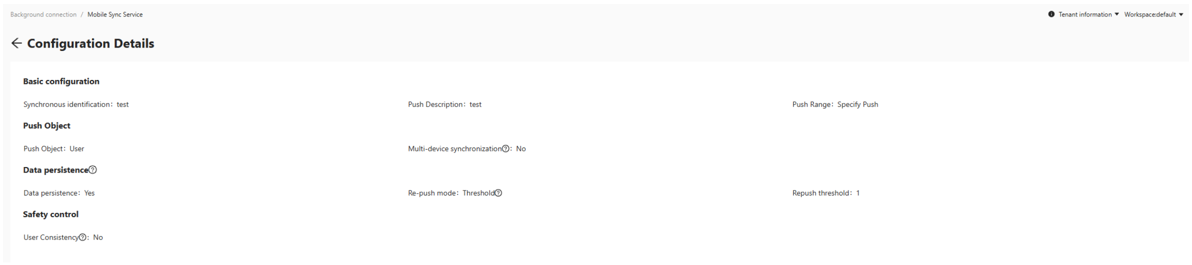
| | |
|---|---|
| Validity period | The maximum validity period for the data in this push. The default value is 30 days. |

## 6.4. View configuration details

In the **mPaaS** console, select the target application, and then follow these steps to view the configuration details.

### Procedure

1. In the navigation pane on the left, click **Background connection** > **Mobile Sync Service**.
2. On the **Configuration Management** tab, click a configuration's **Identification** to view its details.



Background connection / Mobile Sync Service
Tenant information ▾ Workspace:default ▾

← Configuration Details

**Basic configuration**

Synchronous identification: test
Push Description: test
Push Range: Specify Push

**Push Object**

Push Object: User
Multi-device synchronization: No

**Data persistence**

Data persistence: Yes
Re-push mode: Threshold
Repush threshold: 1

**Safety control**

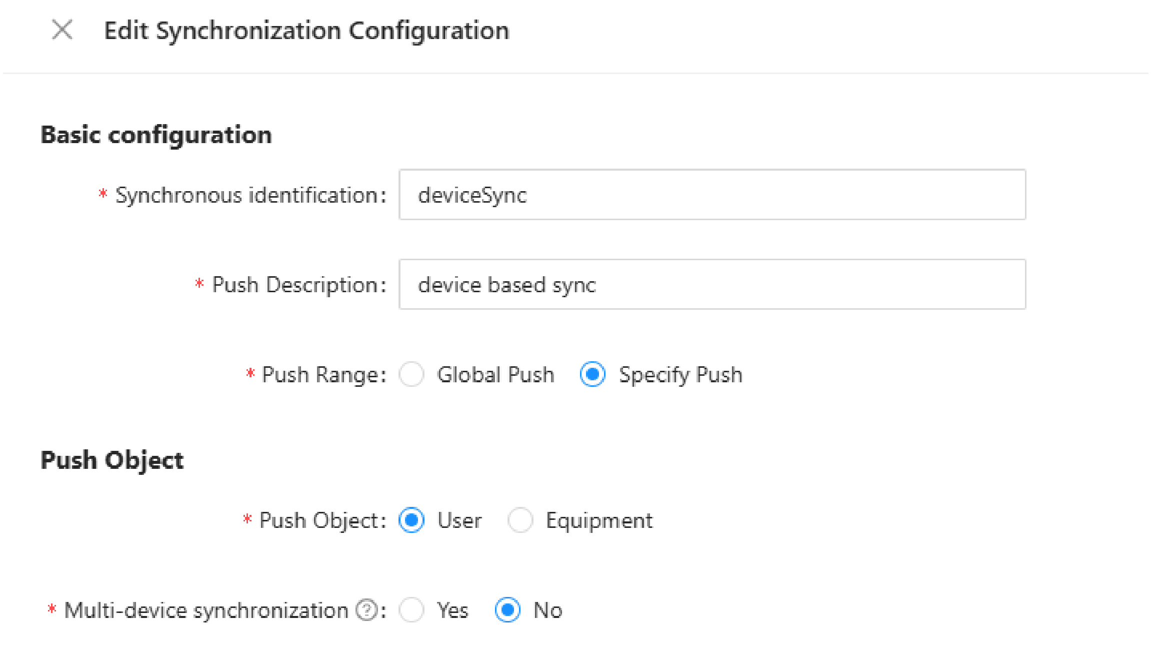
User Consistency: No

## 6.5. Modify configuration

Go to the **mPaaS** console, select the target application, and follow these steps to modify the configuration.

### Procedure

1. In the navigation pane on the left, click **Background connection** > **Mobile Sync Service**.
2. On the **Configuration Management** tab, click **Edit** in the Actions column.
3. In the **Edit Sync Configuration** slide-out panel, modify the configuration.



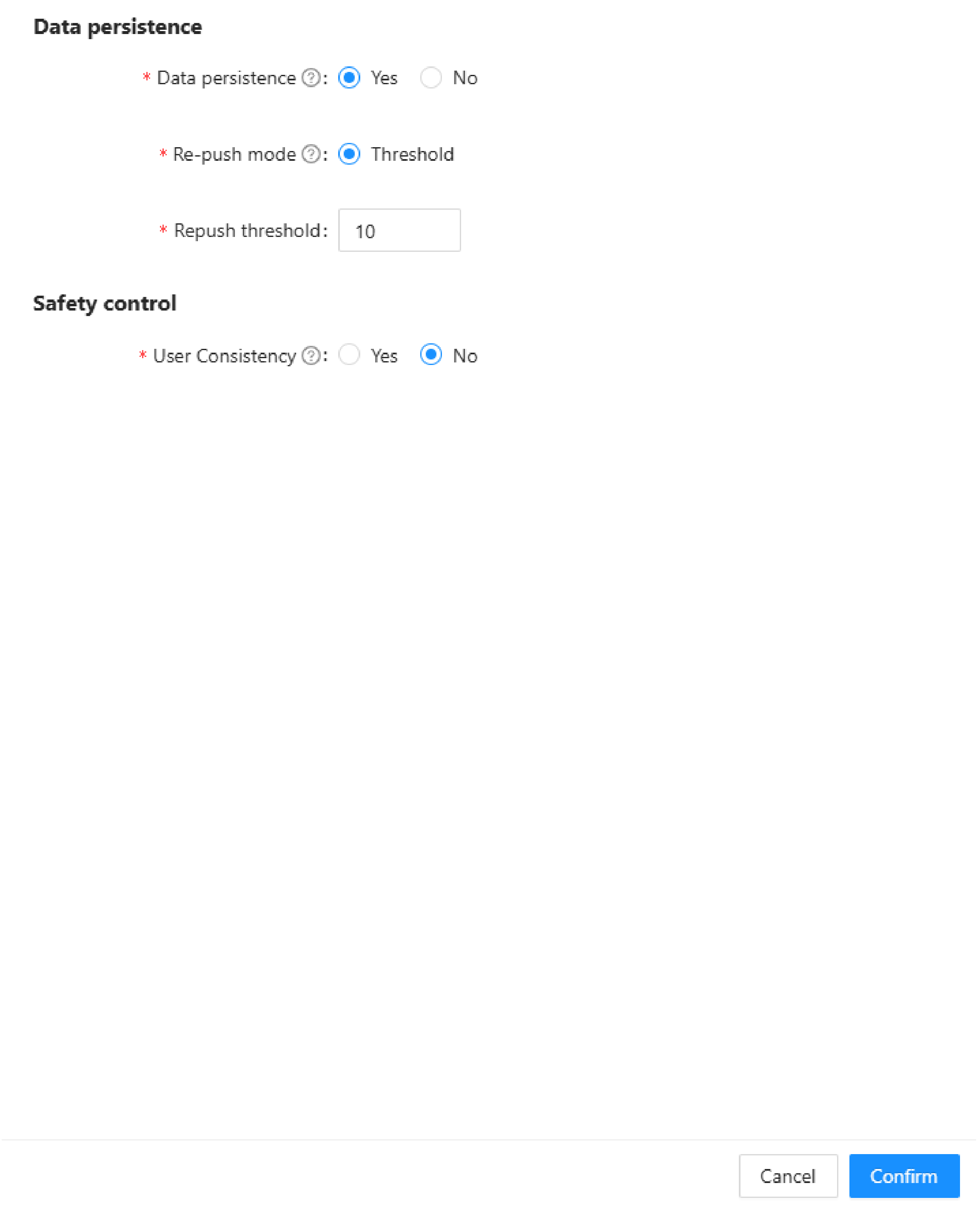
Edit Synchronization Configuration

**Basic configuration**

* Synchronous identification: deviceSync
* Push Description: device based sync
* Push Range: Global Push [✓ Specify Push]

**Push Object**

* Push Object: [✓ User] Equipment
* Multi-device synchronization: Yes [✓ No]


Data persistence

* Data persistence: [✓ Yes] No

* Re-push mode: [✓ Threshold]

* Repush threshold: 10

Safety control

* User Consistency: Yes [✓ No]

## 6.6. Offline configuration

During data synchronization, if you need to pause the process (for example, to urgently stop synchronization due to data issues), you can deactivate the sync configuration. Go to the **mPaaS** console, select the target application, and follow these steps to deactivate the configuration.

### Procedure

1. In the navigation pane on the left, click **Background connection** > **Mobile Sync Service**.

2. On the **Configuration Management** tab, find the target configuration in the list, click **Offline** in the Actions column, and then confirm the operation.

> **Note**
>
> After deactivating a synchronization configuration, the corresponding synchronization task will be temporarily disabled. To use it again, click **Go online** to bring the configuration back online.

## 6.7. Query statistics for configuration pushes

MSS provides push statistics for users and equipments. Log in to the **Mobile PaaS** console, select your target application, and follow these steps to view the statistics for each configuration push.

### Procedure

1. In the navigation pane on the left, click **Background connection > Mobile Sync Service** to go to the Data Synchronization page.
2. On the **Data Query** tab, you can view the status of a user or equipment.
3. In the upper-right corner of the **User/Device Status Query** area, select **User** or **Equipment** from the drop-down list. Enter a username or equipment name in the search box to view the status of a specific user or equipment.

   MSS provides the following data for users or equipments on this page:

   - User/Equipment name
   - Status (Indicates whether the user is connected to the MSS service)
   - Push times in recent 30 days
   - Number of push arrivals in the last 30 days
   - Push list

## 6.8. Service management

The Service Management tab includes a global switch that you can use to temporarily enable or disable all signature verification features.

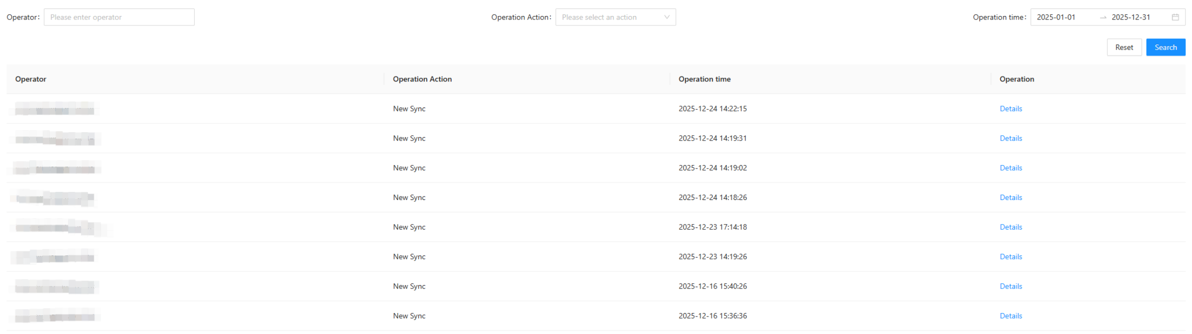
## 6.9. Query MSS operation records

Mobile Sync Service (MSS) allows you to query operation records of data synchronization, including the operation change records of creating a configuration, modifying a configuration, deleting a configuration, creating a synchronization, publishing a configuration, disabling a configuration, enabling a signature, and disabling a signature. MSS is used for operation traceability and usage data analysis.

To query the operation history:

1. Log on to the mPaaS console and select your application. In the navigation pane on the left, click **Background connection > Mobile Sync Service**.

2. On the **Operation record** tab, filter the records by operator account, action type, or date. After you set the filter conditions, click **Search**. The results appear in the list in reverse chronological order. The list shows the operator account, operation type, and exact time (to the second) for each operation.


Operator: Please enter operator
Operation Action: Please select an action
Operation time: 2025-01-01 → 2025-12-31
Reset
Search

| Operator | Operation Action | Operation time | Operation |
|---|---|---|---|
| [redacted] | New Sync | 2025-12-24 14:22:15 | Details |
| [redacted] | New Sync | 2025-12-24 14:19:31 | Details |
| [redacted] | New Sync | 2025-12-24 14:19:02 | Details |
| [redacted] | New Sync | 2025-12-24 14:18:26 | Details |
| [redacted] | New Sync | 2025-12-23 17:14:18 | Details |
| [redacted] | New Sync | 2025-12-23 14:19:26 | Details |
| [redacted] | New Sync | 2025-12-16 15:40:26 | Details |
| [redacted] | New Sync | 2025-12-16 15:36:36 | Details |

**Note**

If you do not specify a time range, the history for the last 7 days is displayed by default. A maximum of 1,000 records are displayed.

3. Find the operation you want to inspect. Click the **Details** link on the right to view its details.
   - **New Configuration**: Shows the details of the new synchronization configuration.
   - **Modify Configuration**: Shows the modified configuration information. The modified content is highlighted in red.
   - **Delete Configuration**: Shows the details of the deleted synchronization configuration.
   - **New Sync**: shows the details of the new synchronization task, including push object ID, data content, unique data ID, app version interval of the client, and push data validity.
   - **Online Configuration**: Displays the details of a synchronization configuration that has been deactivated and then reactivated. The change details shown are the same as those for a newly created configuration.
   - **Offline Configuration**: Shows the details of the unpublished synchronization configuration.
   - **Close Signature**: Shows the record for the disable signing operation.
   - **Open Signature**: Shows the record for the enable signing operation.

# 7.API reference

## 7.1. Android API

**Important**: Since June 28, 2020, mPaaS has stopped support for the baseline 10.1.32. Please use 10.1.68 or 10.1.60 instead. For how to upgrade the baseline from version 10.1.32 to 10.1.68 or 10.1.60, see mPaaS 10.1.68 upgrade guide or mPaaS 10.1.60 upgrade guide.

In baseline 10.1.32 or later versions, the `MPSync` class in the mPaaS middle layer encapsulates all APIs of the Mobile Sync Service (MSS). You can use the `MPSync` object to implement all functions of MSS.

```
java.lang.Object
  - com.mpaas.mss.adapter.api.MPSync
```

Related public functions are shown as follows:

- setup(Application application)
- appToBackground()
- appToForeground()
- clearUserInfo()
- initialize(Context context)
- isConnected()
- registerBiz(String bizType, ISyncCallback syncCallback)
- reportMsgReceived(SyncMessage syncMessag)
- unregisterBiz(String bizType)
- updateUserInfo(String sessionId)

| Return value type | Methods and description |
|---|---|
| void | setup(Application application)<br>Initializes basic services on which MSS depends. Call this API before you call the `initialize` method. This function is available only in baseline 10.1.60 and later versions. |
| void | appToBackground()<br>Notifies the client SDK that the App has been switched to the background and it needs to disconnect from the server. Call this function every time the App is switched to the background. |
| void | appToForeground()<br>Notifies the client SDK that the App has been switched to the foreground and it needs to connect to the server. Call this function every time the App is switched to the foreground. |

| Return value type | Methods and description |
|---|---|
| void | clearUserInfo()<br>Clears user information when a user logs out. |
| void | initialize(Context context)<br>Initializes MSS. |
| boolean | isConnected()<br>Checks whether MSS is running properly. |
| void | registerBiz(String bizType, ISyncCallback syncCallback)<br>Registers a callback to receive business data. If this API is called, the client SDK will call the syncCallback class after receiving synchronized data. |
| void | reportMsgReceived(SyncMessage syncMessag)<br>Notifies MSS of the data synchronization success after data is received in the syncCallback class. Before receiving reportMsgReceived, MSS attempts to resend the data for a maximum of six times. If all resending attempts fail, the data is permanently deleted. |
| void | unregisterBiz(String bizType)<br>Unregisters a specified synchronization configuration. If this API is called, the client SDK will not call the syncCallback class when receiving synchronized data. |
| boolean | updateUserInfo(String sessionId)<br>Call this API at least once when the login information (userId or sessionId) is modified. |

## setup(Application application)

### Declaration

```
public static void setup(Application application)
```

### Description

Used to initialize the base service that MSS depends on. This function needs to be called before the initialize method is called. This function is available only in baseline 10.1.60 and later versions.

### Parameters

| Parameter | Type | Description |
|---|---|---|
| application | Application | An application instance. |

### Returned value

None.

## appToBackground()

### Declaration

```
public static void appToBackground()
```

### Description

Notifies the client SDK that the App has been switched to the background and it needs to disconnect from the server. Call this function every time the App is switched to the background.

We recommend that you call this API inside the `onStop()` method of the home page. If this API is not called when the App is switched to the background, the network connection between the App and the server cannot be released in a timely manner, increasing power consumption and traffic usage.

### Parameters

None.

### Returned value

None.

## appToForeground()

### Declaration

```
public static void appToForeground()
```

### Description

Notifies the client SDK that the App has been switched to the foreground and it needs to connect to the server. Call this function every time the App is switched to the foreground.

We recommend that you call this API inside the `onResume()` method of the home page.

### Parameters

None.

### Returned value

None.

## clearUserInfo()

### Declaration

```
public static void clearUserInfo()
```

### Description

Clears user information when a user logs off.

### Parameters

None.

### Returned value

None.

## initialize(Context context)

### Declaration

```
public static void initialize(Context ctx)
```

### Description

You can call this API to initialize MSS. Your App can use MSS only after you call this API.

During the life cycle of the App (from the time the App is started to the time the App is stopped), this API needs to be called only once.

### Parameters

| Parameter | Type | Description |
|---|---|---|
| ctx | Context | A non-empty `Context`. |

### Returned value

None.

## isConnected()

### Declaration

```
public static boolean isConnected()
```

### Description

Checks whether MSS is running properly.

### Parameters

None.

### Returned value

Returns true if the service is normal, and returns false if the service is abnormal.

## registerBiz(String bizType, ISyncCallback syncCallback)

### Declaration

```
public static void registerBiz(String biz, ISyncCallback callback)
```

### Description

Used to register a callback for receiving service data. If this API is called, the client SDK will call the syncCallback class after receiving synchronized data.

This API needs to be called once for each synchronization configuration.

### Parameters

| Parameter | Type | Description |
|---|---|---|
| bizType | String | Synchronization identifier |
| syncCallback | ISyncCallback | Callback implementation class |

### Returned value

None.

## reportMsgReceived(SyncMessage syncMessag)

### Declaration

```
public static void reportMsgReceived(SyncMessage msg)
```

### Description

After the synchronously pushed data is received in `syncCallback` , call this API to notify MSS that the synchronized data has been received successfully. Before receiving the `reportMsgReceived` , MSS attempts to resend the data for a maximum of six times. If all resending attempts fail, the data will be permanently deleted.

### Parameters

| Parameter | Type | Description |
|---|---|---|
| syncMessag | SyncMessage | Message synchronization |

### Returned value

None.

## unregisterBiz(String bizType)

### Declaration

```
public static void unregisterBiz(String biz)
```

### Description

Unregisters a specified synchronization configuration. MSS will not call `syncCallback` after MSS receives the synchronization configuration data.

### Parameters

| Parameter | Type | Description |
|---|---|---|
| biz | String | Synchronization identifier |

### Returned value

None.

### updateUserInfo(String sessionId)

#### Declaration

`public static boolean updateUserInfo(String sessionId)`

#### Description

Calling inside the method is based on the `LongLinkSyncService.getInstance().updateUserInfo(String userId, String sessionId)` API, in which `userId` indicates the user ID specified in `MPLogger` .This API is called when `userId` or `sessionId` changes and will update user login information.

Both parameters are required for logon. If `userId` is not specified, this method returns `false` , indicating a calling failure.

This method must be called upon session expiration or each successful automatic logon. Note that the automatic logon function is enabled after a user logs on to the client once. The general calling principle is that this method must be called when `userId` or `sessionId` changes.

#### Parameters

| Parameter | Type | Description |
|---|---|---|
| sessionId | String | Session ID. |

#### Returned value

Returns true if the user information is updated successfully, and returns false if userId is not set at logon.

## 7.2. iOS API

The `MPSyncInterface` class in `MPMssAdapter.framework` provides all MSS APIs. All methods in the class are class methods that can be called by the class name.

### +(void)initSync;

Initializes MSS. An app can use MSS only after calling this API.

During the life cycle of the app (from the time the app is started to the time the app is stopped), this API needs to be called only once.

### +(MPSyncNetConnectType)connectStatus;

Checks the connection status of MSS.

Return value: connection status specified by `MPSyncNetConnectType` .

### +(BOOL)registerSyncBizWithName:(NSString *)bizName syncObserver:(id)observer selector:(SEL)selector;

Registers the notification listener which works on the business name `bizName` , and calls `[[NSNotificationCenter defaultCenter] addObserver:observer selector:selector name:bizName object:nil];` to listen on notifications.

The value of **bizName** is the same as that in the server console. If this API is not called, the specified biz messages will not be distributed but stacked in the database of the client SDK. We recommend that you start listening on specified sync messages sent to the server upon server startup.

Return value: registration result `YES` or `NO` .

## +(BOOL)unRegisterSyncBizWithName:(NSString *)bizName syncObserver:(id)observer;

Notifies the MSS client SDK that message listening on a synchronization configuration has been disabled and that sync messages related to the synchronization configuration will no longer be received.

The internal `[[NSNotificationCenter defaultCenter] removeObserver:observer name:bizName object:nil];` API is called to remove the listener.

After this API is called, messages of the biz will not be distributed but stacked in the SyncSDK database. This API matches the `registerSyncBizWithName` API.

Return value: result `YES` or `NO` .

## +(void)removeSyncNotificationObserver:(id)observer;

Disables listening on the synchronization notification. This API is usually called in the `dealloc` function of a listening class. The internal `[[ NSNotificationCenter defaultCenter] removeObserver:observer];` API is called to remove the listener.

Return value: none.

## +(void)responseMessageNotify:(NSDictionary *)userInfo;

Notify a `callback` after a message has been processed. The parameter is `userInfo(notify.userInfo)` in the notification.

Calls back `SyncSDK` , indicating that the business data has been processed in the notification processing function registered using the `registerSyncBizWithName` API, when data processing is completed.

Return value: none.

## +(void)bindUserWithSessionId:(NSString *)sessionId;

This method is called when the value of the login parameter `userId` or `sessionId` changes.

This API is called during login. The value of `userId` is the `-(NSString\*)userId` function of `MPaaSInterface` .

This method must be called upon `sessionId` expiration or each successful automatic login, which is enabled after a user logs in to the client once.

The overall calling principle is that this method must be called when the value of `userId` or `sessionId` changes.

When the value of `userId` changes, `unBindUser` is called to unbind the user account and then `bindUserWithSessionId:` is called to rebuild a connection.

**sessionId** is used with the server to verify the validity of a session. If this parameter is set to **nil** on the server, the default value `@"SESSION_DEMO"` is used.

Return value: none.

## +(void)unBindUser;

Called to unbind the currently connected user when the user logs out.

Return value: none.

### +(NSString *)getSyncDeviceId;

Obtains the device ID, which is used when pushing device-based sync data.

Return value: device ID.

**Important**: If the value of `sessionId` in the API is invalid, the user consistency option in the console must be disabled, or sync messages will fail to be pushed due to verification failure. Enable or disable signature verification by referring to Service management.