

OneAccess

Development Guide

Issue	01
Date	2025-07-16



Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2025. 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

1 Synchronizing Data to Applications Through Event Callback..... 1

1.1 Preparations..... 1

1.2 Calling Methods..... 10

1.2.1 API Calling.....10

1.2.2 Signature Verification.....12

1.2.3 Verifying Callback URL..... 20

1.3 API..... 22

1.3.1 Overview.....22

1.3.2 Adding an Organization Event.....22

1.3.3 Modifying an Organization Event.....23

1.3.4 Deleting an Organization Event..... 25

1.3.5 Adding a User Event..... 27

1.3.6 Modifying a User Event..... 29

1.3.7 Deleting a User Event..... 31

1.4 Common Return Codes..... 32

2 Developing Mapping Scripts.....33

Issue 01 (2025-07-16)

Copyright © Huawei Cloud Computing Technologies Co., Ltd.

ii

1 Synchronizing Data to Applications Through Event Callback

1.1 Preparations

OneAccess synchronizes identity data using the "Upstream – Midstream – Downstream" model. Upstream refers to various core identity sources, such as DingTalk, WeCom, HR, and OneAccess, midstream is OneAccess, and downstream indicates an application system that synchronizes identity data with the upstream. In this model, OneAccess transfers real-time identity data from upstream to downstream, maintaining consistency and security of identity data throughout the user lifecycle, covering onboarding, job transfer, and resignation.

To synchronize OneAccess data to enterprise applications, you need to develop an event synchronization API based on the data format in advance. For details, see [Calling Methods](#) and [API](#).

The following describes how to synchronize OneAccess data to an application through event callback:

1. [Event Callback Configuration](#)
2. [Adding, Modifying, and Deleting Users](#)
3. [Adding, Modifying, and Deleting Organizations](#)
4. [Event Synchronization](#)
5. [Full Synchronization](#)
6. [Synchronization Statuses](#)

Event Callback Configuration

As an administrator, you can configure event callback on the administrator portal.

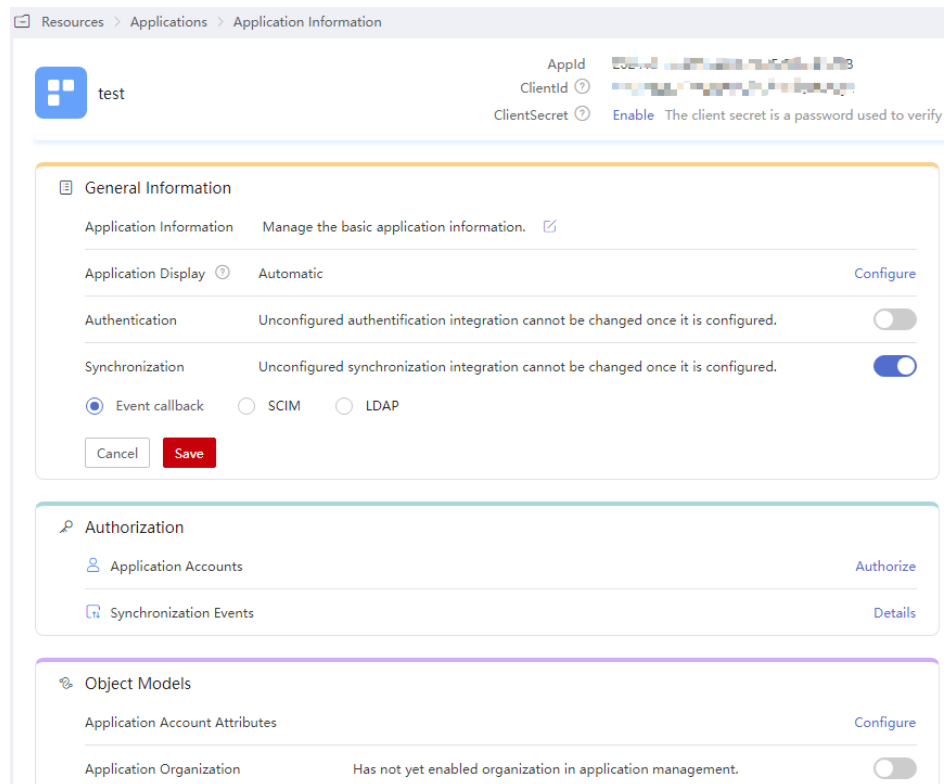
Step 1 Log in to the administrator portal.

Step 2 On the top navigation bar, choose **Resources > Applications**.

Step 3 Click **Add Custom Application** in the **Custom Applications** section, set the logo and application name, and click **Save**.

Step 4 Click the application created in [Step 3](#). In the **General Information** area, click  next to **Synchronization** and select **Event callback**. Click **Save**.

Figure 1-1 Enabling event callback



Step 5 In the **General Information** area, click **Configure** next to **Synchronization**. On the **Parameters** tab, set parameters according to [Table 1-1](#) and click **Save**.

Table 1-1 Parameters

Parameter	Description
Callback URL	URL used by enterprise applications to receive events pushed by OneAccess.
Security Token	The request header of each event callback API carries a bear token. The callback service of the enterprise application needs to be authenticated.
Encryption/Decryption Algorithm	AES/GCM/NOPadding (default) algorithms are recommended. NULL indicates that data is not encrypted, which has security risks. Exercise caution when using this method.
Encryption Key	Keys used to encrypt a message. The value is left blank by default. Otherwise, the value must contain 16 digits.

Parameter	Description
Signature Key	Signature key used to generate a data signature based on the message content. The value is left blank by default. Otherwise, the value must contain 16 digits.

Step 6 (Optional) Click the application icon. In the navigation pane, choose **Synchronization Integration** and select the **General** tab to modify the mapping during synchronization, including deleting and deactivating accounts and organizations.

Step 7 (Optional) Click the application icon. In the navigation pane, choose **Object Models > Application Organization** and click  to enable the application organization.

 NOTE

Skip this step if you will not synchronize organization data to the application.

Step 8 (Optional) Configure organization attributes and mappings. Skip this step if you only need to synchronize the built-in attributes of the organization.

1. In the navigation pane, choose **Object Models > Application Organization**. On the **Attributes** tab, click **Add** to configure the organization attributes synchronized to enterprise applications.

 NOTE

- When synchronizing built-in attributes to enterprise applications, ensure that the added attribute names are the same as those in application organizations.
- When synchronizing non-built-in attributes to enterprise applications, ensure that the attribute names are the same as those set by administrators.

Table 1-2 Attribute parameters

Parameter	Description
Attribute (Mandatory)	Custom field ID synchronized to enterprise applications, which cannot be changed after being set.
Label (Mandatory)	Attribute name of an organization. It is recommended that the value be the same as that of Attribute .
Description	Description of the Attribute .
Attribute Type (mandatory)	Type of the Attribute . You can select a value from the drop-down list. The value cannot be changed after being set.

Parameter	Description
Format	When Attribute Type is set to Text , you can select a value from the drop-down list.
Required	This option is not selected by default.
Unique	When Attribute Type is set to Text , you can select Unique . If this option is selected, the attribute value must be unique when organization data is synchronized to the application. If the attribute value is duplicate, a message is displayed, indicating that the label already exists.
Sensitive	It is required only when Attribute Type is set to Text . If this option is selected, the application data is hidden when it is synchronized to an application. You can click  to view the content.

2. On the **Mappings** tab, click **Modify** and set the mappings of the organization attribute.

 NOTE

- When **Conversion Mode** is set to **Script-based**, compile the script by referring to [Developing Mapping Scripts](#).
- To prevent synchronization exceptions, you are advised to add organization attributes of the same type as the application organization attributes to be mapped.

Table 1-3 Mapping parameters

Parameter	Description
Organization	Organization attribute mapped to the Application Organization . You can select a value from the drop-down list.
Conversion Mode	Mode of attribute mapping between Organization and Application Organization .
Script Expression	When Conversion Mode is set to Script-based , enter the specific mapping script.
Execution Mode	Mode of attribute mapping between Organization and Application Organization .
Application Organization	Organization attributes defined in 1 .

- On the **Configure** tab, **Delete System Org** is set to **Delete application organization** and **Disable System Org** is set to **Disable application organization** by default. You can click **Modify** to modify the configuration. You can set **Delete System Org** to **Disable application organization** or **Do not affect**, and set **Disable System Org** to **Do not affect**. Click **Save** for the modification to take effect.

Step 9 (Optional) Configure account attributes and mappings. Skip this step if you only need to synchronize the built-in attributes of the account.

- In the navigation pane, choose **Object Models > Application Accounts**. On the **Attributes** tab, click **Add** to configure account attributes synchronized to enterprise applications.

 NOTE

- When synchronizing built-in attributes to enterprise applications, ensure that the added attribute names are the same as those in application accounts.
- When synchronizing non-built-in attributes to enterprise applications, ensure that the attribute names are the same as those set by administrators.

Table 1-4 Attribute parameters

Parameter	Description
Attribute (Mandatory)	Custom field ID synchronized to enterprise applications, which cannot be changed after being set.
Label (mandatory)	Attribute name of an account. It is recommended that the value be the same as that of Attribute .
Description	Description of the Attribute .
Attribute Type (mandatory)	Type of the Attribute . You can select a value from the drop-down list. The value cannot be changed after being set.
Format	When Attribute Type is set to Text , you can select a value from the drop-down list.
Required	This option is not selected by default.
Unique	When Attribute Type is set to Text , you can select Unique . If this option is selected, the attribute value must be unique when user data is synchronized to the application. If the attribute value is duplicate, a message is displayed, indicating that the label already exists.

Parameter	Description
Sensitive	It is required only when Attribute Type is set to Text . If this option is selected, the user data is hidden when it is synchronized to an application. You can click  to view the content.

2. On the **Mappings** tab, click **Modify** and set the mappings of the organization attribute.

 NOTE

- When **Conversion Mode** is set to **Script-based**, compile the script by referring to [Developing Mapping Scripts](#).
- To prevent synchronization exceptions, you are advised to add account attributes of the same type as the user attributes to be mapped.

Table 1-5 Mapping parameters

Parameter	Description
User	User attribute mapped to the Application Accounts . You can select a value from the drop-down list.
Conversion Mode	Mode of attribute mapping between User and Application Accounts .
Script Expression	When Conversion Mode is set to Script-based , enter the specific mapping script.
Execution Mode	Mode of attribute mapping between User and Application Accounts .
Application Accounts	Account attributes defined in 1 .

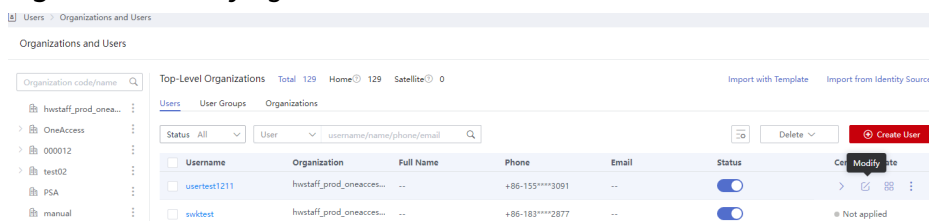
3. On the **Configure** tab, **Delete System User** is set to **Delete application account** and **Disable System User** is set to **Disable application account** by default. Click **Modify** to modify the configuration. If you select to disable or retain the application account for **Delete System User**, the account automatically changes to an orphan account because the user has been deleted. You can select to retain an application account for **Disable System User**. Click **Save** for the modification to take effect.

----End

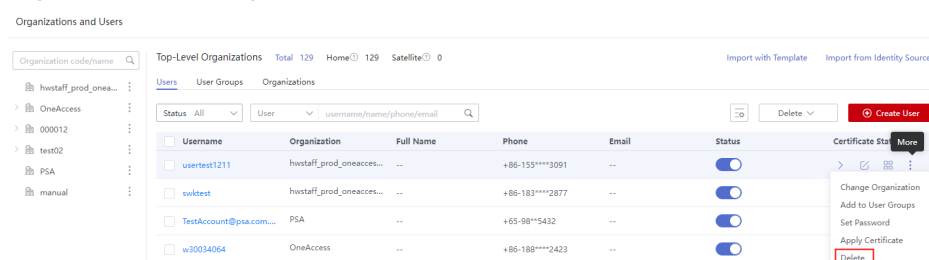
Adding, Modifying, and Deleting Users

As an administrator, you can synchronize users on the administrator portal.

- Adding users
In the navigation pane, choose **Authorization > Application Accounts**. Then click **Add Accounts** to authorize specific accounts to access the application. To synchronize accounts based on the authorization policy, see [Configuring Authorization Policy for Application Accounts](#).
- Modifying users
Choose **Users > Organizations and Users** from the top navigation bar. In the user list, move the cursor to the right of the target username and click . In the **Modify User** dialog box, update the user information.

Figure 1-2 Modifying user information

- Deleting users
Choose **Users > Organizations and Users** from the top navigation bar. In the user list, click  next to the target username, select **Delete**, and click **OK**.

Figure 1-3 Deleting users

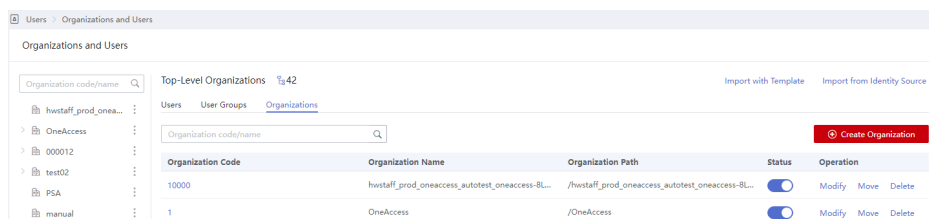
Adding, Modifying, and Deleting Organizations

As an administrator, you can synchronize organizations on the administrator portal.

NOTE

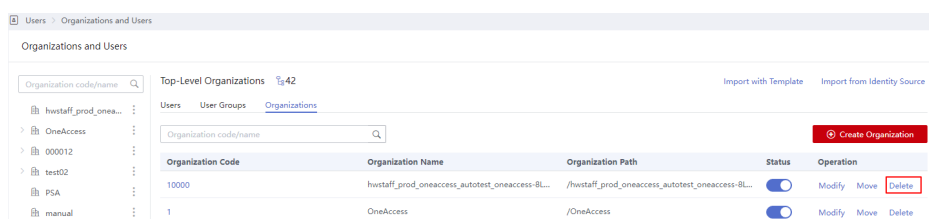
Enable the application synchronization before synchronizing organizations. For details, see [Step 7](#).

- Adding organizations
In the navigation pane, choose **Authorization > Application Organizations**, click **Authorization Policy**, and click  to enable **Automatic Organization Authorization**. Select the organizations to be synchronized as prompted and click **Save**.
- Modifying organizations
Choose **Users > Organizations and Users** from the top navigation bar. On the **Organizations** page, click **Modify** in the **Operation** column of the target organization to update the organization information.

Figure 1-4 Modifying an organization

- Deleting an organization

Choose **Users > Organizations and Users** from the top navigation bar. On the **Organizations** page, click **Delete** in the **Operation** column of the target organization to delete it.

Figure 1-5 Deleting an organization

Event Synchronization

When OneAccess synchronizes data to downstream applications, all synchronization operations are recorded.

1. Log in to the administrator portal.
2. On the top navigation bar, choose **Resources > Applications**.
3. On the displayed page, click an application name to access the application details page.
4. Click the application icon to access the general information page.
5. In the navigation pane, choose **Authorization > Synchronization Events** to access the page. You can filter synchronization records by time, operation type, object type, and synchronization status.

NOTE

If synchronization failed:

- Check the response and rectify the fault, and click **Retry** in the **Operation** column.
- Click **Retry** to quickly perform synchronization. After you successfully retry the synchronization event of the parent organization, the synchronization events of the sub-organizations and accounts under the parent organization will be triggered.

Full Synchronization

Full synchronization can be used to ensure full data consistency between downstream application systems and OneAccess. Full synchronization complies with the following rules:

- During full synchronization, all previous synchronization events are ignored, and new events are generated. If the application system does not return an

ID, a new event is generated. However, if the application system returns an ID, an update event is generated. For details, see [Table 1-12](#).

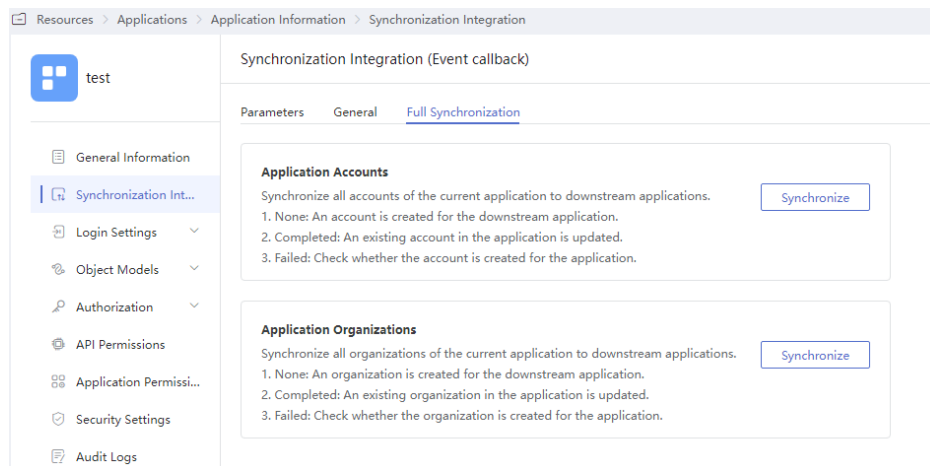
NOTE

An update event triggered by full synchronization synchronizes all attributes configured in mappings. A common update event synchronizes only changed attributes.

- If the dependent organization fails to be synchronized, all synchronization events of its sub-organizations and accounts are suspended. In this case, click **Synchronize** next to **Application Organizations** to trigger a full synchronization of application accounts. This will also activate **Synchronize** next to **Application Accounts**.

As an administrator, you can synchronize all data to application systems on the administrator portal.

In the navigation pane, choose **Synchronization Integration**, select **Full Synchronization** tab, and click **Synchronize** either next to **Application Accounts** or **Application Organizations**.



Synchronization Statuses

This section describes the different statuses after the synchronization is complete. This helps you locate faults.

The following is an example, in which **App_acc01** is the application account, **App_org01** application organization, **E0** original synchronization event, and **E1** new synchronization event.

- **Pending**
When the application account **App_acc01** generates a new synchronization event **E1**, if the execution of a previous synchronization event **E0** for the same account is not yet completed, the status of the new event **E1** is **PENDING**.
- **Queuing**
When the synchronization event **E1** for the application account **App_acc01** is ready to be executed, the event is sent to the Kafka execution message queue. Once the event is successfully sent, the status of synchronization event **E1** in ES is updated to **QUEUING**, indicating that the event has been successfully added to the execution queue.

- Running
When the synchronization event **E1** is ready to execute, its status in ES is updated to **RUNNING** before the execution starts. Once the status is successfully updated, the downstream application system API is triggered to start synchronization, indicating that the event is now in the running status.
- Success
After a synchronization event is successfully executed, its status is updated to **SUCCESS** in ES, indicating that the event has been successfully executed.
- Failure
After a synchronization event fails to execute, its status is updated to **FAILURE** in ES, indicating that the event failed to be executed.
- Ignored
This status involves the following two scenarios:
 - When the application account **App_acc01** executes an update synchronization event, if a new update event is found subsequently, this event is directly ignored, and the new update event will be executed instead.
 - During full synchronization, all previous synchronization events are ignored.
- Skipped
This status is not used currently.
- Waiting
When the application account **App_acc01** has a dependency on the application organization **App_org01**, if the synchronization of **App_org01** fails, all synchronization events of **App_acc01** in the organization are suspended.

1.2 Calling Methods

1.2.1 API Calling

Format

The request method of the OneAccess synchronous event callback API is POST, with data encoded in UTF-8 and formatted as JSON. If the URL for the application system to receive event callbacks is **https://{app_domain}/callback**, OneAccess will push the updated service data to this callback address whenever there are changes in the organizations or users.

- URL
POST https://{app_domain}/callback
- Request header
Authorization: **Bearer** {access_token}. For details, see security tokens in [Table 1-1](#).
- Request parameters

Table 1-6 Request parameters

Parameter	Type	Description
nonce	String	Random number, which is used together with timestamp to prevent replay attacks on requests.
timestamp	Integer	Timestamp, which is used together with nonce to prevent replay attacks on requests.
eventType	String	Event type. For details, see the event type list.
data	String	Message body. If encryption is disabled, the message body is sent in plaintext. If encryption is enabled, the message body must be decrypted to reveal the content. After decryption, the random and msg fields will be displayed. The msg field contains the plaintext message content.
signature	String	Message signature. If the signature function is disabled, the signature will be an empty string. If the signature function is enabled, the signature is generated based on the signature key (signatureSaltValue) provided by the enterprise application, along with the timestamp , nonce from the request, and the encrypted message body.

- Response parameters

Table 1-7 Response parameters

Parameter	Type	Description
code	String	Return code. The value 200 indicates success. For details about error codes, see Common Return Codes .
message	String	Description of the error cause.
data	String	Returned message body. The returned content varies depending on the service callback. For example, it may return an empty string or the required service data. • If encryption is disabled, the message body is returned in plaintext. • If encryption is enabled, the encrypted message body is returned. The content must be decrypted, after which the random and msg fields are generated. The msg field contains the plaintext message content.

- Example request
 - Example request with message signature and encryption disabled:

```
{
  "nonce": "123456",
  "timestamp": 1783610513,
  "eventType": "eventType",
  "data": "plaintext message",
  "signature": ""
}
```

- Example request with message signature and encryption enabled:

```
{
  "nonce": "123456",
  "timestamp": 1783610513,
  "eventType": "eventType",
  "data": "1ojvw2WPvW7LijxS8UvISr8pdDP+rXpPbcLGOmIBNbWetRg7IP0vdhkl",
  "signature": "111108bb8e6dbce3c9671d6fdb69d15066227608"
}
```

- Example response

Status code: 200

Request successful.

- Response example with message signature and encryption disabled:

```
{
  "code": "200",
  "message": "success",
  "data": "plaintext message"
}
```

- Response example with message signature and encryption enabled:

```
{
  "code": "200",
  "message": "success",
  "data": "P+rXpWetRg7IP0vdhVgkVwSoZBJeQwY2zhROsJq/HJ+q6tp1qhl9L1+c"
}
```

1.2.2 Signature Verification

When OneAccess synchronizes data to an enterprise application, the application must identify and confirm the synchronization event to check the security and reliability of the event source and to safeguard that data is exchanged in a secure environment.

Signature Verification/Encryption and Decryption Terms

Table 1-8 Terms

Term	Description
signature	Message signature, which is used to verify whether a request is from OneAccess to prevent attackers from forging the request. The signature algorithm is HMAC-SHA256 + Base64 .
AESKey	Key of the AES algorithm. The encryption algorithm is AES/GCM/NoPadding + Base64 .
msg	Plaintext message body in JSON format.
encrypt_msg	Encrypted and Base64-encoded ciphertext of the plaintext message 'msg'.

Signature Verification

To ensure that the event is pushed by OneAccess, the request body includes a request signature, identified by the **signature** parameter, when OneAccess pushes the event to the enterprise application callback service. The enterprise application must verify this parameter before decryption. The verification procedure is as follows:

1. Calculating signatures

The signature consists of the signature key, *nonce*, *timestamp*, *eventType*, and data, all concatenated using ampersands (&). The **HMAC-SHA256 + Base64** algorithms are used for encryption. The following is an example of the Java signature:

```
String message = nonce + "&" + timestamp + "&" + eventType + "&" + data;
Mac mac = Mac.getInstance("HmacSHA256");
SecretKeySpec secretKey = new SecretKeySpec(Signature key.getBytes(StandardCharsets.UTF_8),
"HmacSHA256");
mac.init(secretKey);
String newSignature =
Base64.getEncoder().encodeToString(mac.doFinal(message.getBytes(StandardCharsets.UTF_8)));
```

2. Check whether the calculated signature **cal_signature** is the same as **signature** in the request parameter. If they are the same, the verification is successful.
3. The response message returned by the enterprise application needs to use the required format.

Encrypting a Plaintext

1. Concatenate plaintext strings, which consist of a 16-byte random characters and a plaintext **msg**, and are separated by ampersands (&). The following is an example written in Java:

```
String dataStr = RandomStringUtils.random(16, true, false) + "&" + data;
```

2. Use the AESKey to encrypt the concatenated plaintext string and encode it using Base64 to obtain the ciphertext **encrypt_msg**. The following is an example written in Java:

```
Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
SecretKeySpec secretKey = new SecretKeySpec(Encryption key.getBytes(StandardCharsets.UTF_8),
"AES");
cipher.init(1, secretKey);
byte[] bytes = dataStr.getBytes(StandardCharsets.UTF_8);
String encryptStr = Base64.getEncoder().encodeToString(cipher.doFinal(bytes));
```

Decrypting Ciphertext

1. Decode the ciphertext using Base64.

```
byte[] encryptStr = Base64.getDecoder().decode(data);
```
2. Use the AESKey for decryption.

```
Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
SecretKeySpec secretKey = new SecretKeySpec(Encryption key.getBytes(StandardCharsets.UTF_8),
"AES");
cipher.init(2, secretKey);
byte[] bytes = cipher.doFinal(encryptStr);
```
3. Remove the 16 random bytes from the **rand_msg** header. The remaining part is the plaintext **msg**.

```
String dataStr = StringUtils.split(new String(bytes, StandardCharsets.UTF_8), "&")[1];
```

Examples of Data Signature & Encryption/Decryption

- The following is an example of data signature and AES/GCM/NOPadding encryption and decryption written in Java:

```
package com.example.demo.controller;
import com.alibaba.fastjson.JSONObject;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.util.Base64;
import javax.crypto.Cipher;
import javax.crypto.Mac;
import javax.crypto.spec.GCMParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import javax.servlet.http.HttpServletRequest;
import org.apache.commons.lang3.RandomStringUtils;
import org.apache.commons.lang3.StringUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.util.Assert;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
@Controller
public class GcmController {
    private static final Logger log =
        LoggerFactory.getLogger(com.example.demo.controller.GcmController.class);
    //region. The following three parameters are 32-bit strings randomly generated by a third party
    (case-sensitive letters and digits). Do not use the example in the formal environment.
    /**
     * Security token
     */
    private static String token;
    /**
     * Signature key
     */
    private static String signKey;
    /**
     * Encryption key
     */
    private static String encryptionKey;
    //endregion
    /**
     * Encryption algorithm
     */
    private static String hmacSha256;
    private static String encryptAlgorithm;
    private static String encryptKeyAlgorithm;
    private static int aesKeySize;
    private static final Charset CHARSET = StandardCharsets.UTF_8;
    public static void setToken(String token) {
        GcmController.token = token;
    }
    public static void setSignKey(String signKey) {
        GcmController.signKey = signKey;
    }
    public static void setEncryptionKey(String encryptionKey) {
        GcmController.encryptionKey = encryptionKey;
    }
    public static void setHmacSha256(String hmacSha256) {
        GcmController.hmacSha256 = hmacSha256;
    }
    public static void setEncryptAlgorithm(String encryptAlgorithm) {
        GcmController.encryptAlgorithm = encryptAlgorithm;
    }
    public static void setEncryptKeyAlgorithm(String encryptKeyAlgorithm) {
        GcmController.encryptKeyAlgorithm = encryptKeyAlgorithm;
    }
}
```

```
public static void setAesKeySize(int aesKeySize) {
    GcmController.aesKeySize = aesKeySize;
}
public static String getToken() {
    return token;
}
public static String getSignKey() {
    return signKey;
}
public static String getEncryptionKey() {
    return encryptionKey;
}
public static String getHmacSha256() {
    return hmacSha256;
}
public static String getEncryptAlgorithm() {
    return encryptAlgorithm;
}
public static String getEncryptKeyAlgorithm() {
    return encryptKeyAlgorithm;
}
public static int getAesKeySize() {
    return aesKeySize;
}
@ResponseBody
@RequestMapping("/{gcm/callback}")
public JSONObject demo(HttpServletRequest httpRequest, @RequestBody String body) {
    log.info("Receive event callback information data. Original packet:" + body);
    JSONObject result = new JSONObject();
    result.put("code", "200");
    result.put("message", "success");
    String authorization = httpRequest.getHeader("Authorization");
    if (!StringUtils.join((Object[]) new String[]{"Bearer ", token}).equals(authorization)) {
        result.put("code", "401");
        result.put("message", "Invalid request!");
        printResult(result);
        return result;
    }
    JSONObject request = JSONObject.parseObject(body);
    String signature = request.getString("signature");
    String eventType = request.getString("eventType");
    log.info("Event type" + eventType);
    String data = request.getString("data");
    if (StringUtils.isNotEmpty(signKey))
        try {
            log.info("Start to verify signature");
            String message = request.getString("nonce") + "&" + request.getLong("timestamp") + "&"
+ eventType + "&" + data;
            Mac mac = Mac.getInstance(hmacSha256);
            SecretKeySpec secretKey = new SecretKeySpec(signKey.getBytes(CHARSET), hmacSha256);
            mac.init(secretKey);
            String newSignature =
Base64.getEncoder().encodeToString(mac.doFinal(message.getBytes(CHARSET)));
            Assert.isTrue(newSignature.equals(signature), "Signature inconsistency");
            log.info("Signature verified");
        } catch (Exception e) {
            log.error("Verify signature failed", e);
            result.put("code", "401");
            result.put("message", "Verify signature failed");
            printResult(result);
            return result;
        }
    if (StringUtils.isNotEmpty(encryptionKey)) {
        log.info("Start to decrypt data" + data);
        try {
            Cipher cipher = Cipher.getInstance(encryptAlgorithm);
            SecretKeySpec secretKey = new SecretKeySpec(encryptionKey.getBytes(CHARSET),
encryptKeyAlgorithm);
            byte[] iv = decodeFromBase64(data.substring(0, 24));
```

```
        data = data.substring(24);
        cipher.init(2, secretKey, new GCMParameterSpec(aesKeySize, iv));
        byte[] bytes = cipher.doFinal(Base64.getDecoder().decode(data));
        data = new String(bytes, CHARSET);
        log.info("Data decrypted. Decrypted data:" + data);
    } catch (Exception e) {
        log.error("Decrypt data failed", e);
        result.put("code", "401");
        result.put("message", "Decrypt data failed");
        printResult(result);
        return result;
    }
}
JSONObject eventData = JSONObject.parseObject(data);
JSONObject returnData = new JSONObject(1);
String dataStr = null;
switch (eventType) {
    case "CREATE_USER":
        // Create downstream data based on the unique username. If the downstream data
        already exists, update it and return the downstream ID.
        returnData.put("id", eventData.getString("username"));
        break;
    case "CREATE_ORGANIZATION":
        // Create downstream data based on the unique code. If the downstream data already
        exists, update it and return the downstream ID.
        returnData.put("id", eventData.getString("code"));
        break;
    case "UPDATE_USER":
    case "UPDATE_ORGANIZATION":
        // When data is updated, the fields that are not modified are empty.
        returnData.put("id", eventData.getString("id"));
        break;
    case "DELETE_ORGANIZATION":
    case "DELETE_USER":
        // Perform the deletion operation based on the downstream service logic. No fields need to
        be returned.
        break;
    default:
        result.put("code", "400");
        result.put("message", "Unsupported event type");
        printResult(result);
        return result;
}
if (dataStr == null && returnData.size() > 0)
    dataStr = returnData.toJSONString();
if (StringUtils.isNotEmpty(encryptionKey) && dataStr != null) {
    String random = RandomStringUtils.random(24, true, true);
    log.info("Start to encrypt data" + dataStr);
    try {
        Cipher cipher = Cipher.getInstance(encryptAlgorithm);
        SecretKeySpec secretKey = new SecretKeySpec(encryptionKey.getBytes(CHARSET),
encryptKeyAlgorithm);
        byte[] iv = decodeFromBase64(random);
        cipher.init(1, secretKey, new GCMParameterSpec(aesKeySize, iv));
        byte[] bytes = dataStr.getBytes(CHARSET);
        dataStr = Base64.getEncoder().encodeToString(cipher.doFinal(bytes));
        dataStr = random + dataStr;
        log.info("Data encrypted. Encrypted data:" + dataStr);
    } catch (Exception e) {
        log.error("Encrypt data failed", e);
        result.put("code", "500");
        result.put("message", "Encrypt data failed");
        printResult(result);
        return result;
    }
}
result.put("data", dataStr);
printResult(result);
return result;
```

```
}
private static byte[] decodeFromBase64(String data) {
    return Base64.getDecoder().decode(data);
}
private void printResult(JSONObject result) {
    log.info("'" + result.toJSONString());
}
}
```

- The following is an example of data signature and AES/ECB/PKCS5Padding encryption and decryption written in Java:

```
package com.example.demo.controller;
import com.alibaba.fastjson.JSONObject;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.util.Base64;
import java.util.UUID;
import javax.crypto.Cipher;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import javax.servlet.http.HttpServletRequest;
import org.apache.commons.lang3.RandomStringUtils;
import org.apache.commons.lang3.StringUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.util.Assert;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@RequestMapping
@Controller
public class DemoController {
    private static final Logger log =
        LoggerFactory.getLogger(com.example.demo.controller.DemoController.class);

    /** Signature concatenation operator */
    private static String SEPARATOR = "&";

    /** Signature key */
    private static String signKey;

    /** Encryption key */
    private static String encryptionKey;

    /** Security token */
    private static String token;

    /** Signature algorithm */
    private static String hmacSha256;

    /** Encryption algorithm */
    private static String encryptAlgorithm;

    /** */
    private static String encryptKeyAlgorithm;

    /** Character encoding */
    private static final Charset CHARSET = StandardCharsets.UTF_8;
    public static String getSignKey() {
        return signKey;
    }
    public static String getEncryptionKey() {
        return encryptionKey;
    }
    public static String getToken() {
        return token;
    }
    public static String getHmacSha256() {
```

```
        return hmacSha256;
    }
    public static String getEncryptAlgorithm() {
        return encryptAlgorithm;
    }
    public static String getEncryptKeyAlgorithm() {
        return encryptKeyAlgorithm;
    }
    public static void setSignKey(String signKey) {
        DemoController.signKey = signKey;
    }
    public static void setEncryptionKey(String encryptionKey) {
        DemoController.encryptionKey = encryptionKey;
    }
    public static void setToken(String token) {
        DemoController.token = token;
    }
    public static void setHmacSha256(String hmacSha256) {
        DemoController.hmacSha256 = hmacSha256;
    }
    public static void setEncryptAlgorithm(String encryptAlgorithm) {
        DemoController.encryptAlgorithm = encryptAlgorithm;
    }
    public static void setEncryptKeyAlgorithm(String encryptKeyAlgorithm) {
        DemoController.encryptKeyAlgorithm = encryptKeyAlgorithm;
    }
    @ResponseBody
    @RequestMapping("/{callback}")
    public JSONObject demo(HttpServletRequest httpRequest, @RequestBody String body) {
        JSONObject result = new JSONObject();
        result.put("code", "200");
        result.put("message", "success");
        log.info ("Receive event callback message data. Original packet:" + body);
        String authorization = httpRequest.getHeader("Authorization");
        if (!StringUtils.join("Bearer ", token).equals(authorization)) {
            result.put("code", "401");
            result.put("message", "Invalid request!");
            printResult(result);
            return result;
        }
        JSONObject request = JSONObject.parseObject(body);
        String signature = request.getString("signature");
        String eventType = request.getString("eventType");
        log.info("Event type" + eventType);
        String data = request.getString("data");
        if (StringUtils.isEmpty(signKey)) {
            try {
                log.info("Start to verify signature");
                String message = request.getString("nonce") + SEPARATOR +
request.getLong("timestamp") + SEPARATOR + eventType + SEPARATOR + data;
                Mac mac = Mac.getInstance(hmacSha256);
                SecretKeySpec secretKey = new SecretKeySpec(signKey.getBytes(CHARSET), hmacSha256);
                mac.init(secretKey);
                String newSignature =
Base64.getEncoder().encodeToString(mac.doFinal(message.getBytes(CHARSET)));
                Assert.isTrue(newSignature.equals(signature), "Signature inconsistency");
                log.info("Signature verified");
            } catch (Exception e) {
                log.info ("Verify signature failed");
                log.error ("Verify signature failed", e);
                result.put("code", "401");
                result.put("message", "Verify signature failed");
                printResult(result);
                return result;
            }
        }
        if (StringUtils.isEmpty(encryptionKey)) {
            log.info("Start to decrypt data" + data);
            try {
```

```
Cipher cipher = Cipher.getInstance(encryptAlgorithm);
SecretKeySpec secretKey = new SecretKeySpec(encryptionKey.getBytes(CHARSET),
encryptKeyAlgorithm);
cipher.init(2, secretKey);
byte[] bytes = cipher.doFinal(Base64.getDecoder().decode(data));
data = StringUtils.split(new String(bytes, CHARSET), SEPARATOR)[1];
log.info("Data decrypted. Decrypted data:" + data);
} catch (Exception e) {
    log.info ("Decrypt data failed");
    log.error ("Data decryption error", e);
    result.put("code", "401");
    result.put("message", "Decrypt data failed");
    printResult(result);
    return result;
}
}
JSONObject eventData = JSONObject.parseObject(data);
JSONObject returnData = new JSONObject(1);
String dataStr = null;
switch (eventType) {
    case "CREATE_USER":
        returnData.put("id", eventData.getString("username"));
        break;
    case "CREATE_ORGANIZATION":
        returnData.put("id", eventData.getString("code"));
        break;
    case "UPDATE_USER":
    case "UPDATE_ORGANIZATION":
        returnData.put("id", eventData.getString("id"));
        break;
    case "DELETE_ORGANIZATION":
    case "DELETE_USER":
        break;
    case "CHECK_URL":
        dataStr = UUID.randomUUID().toString().replaceAll("-", "");
        break;
    default:
        result.put("code", "400");
        result.put("message", "Unsupported event type");
        printResult(result);
        return result;
}
if (dataStr == null && returnData.size() > 0) {
    dataStr = returnData.toJSONString();
}
if (StringUtils.isNotEmpty(encryptionKey) && dataStr != null) {
    dataStr = RandomStringUtils.random(16, true, false) + SEPARATOR + dataStr;
    log.info("Start to encrypt data" + dataStr);
    try {
        Cipher cipher = Cipher.getInstance(encryptAlgorithm);
        SecretKeySpec secretKey = new SecretKeySpec(encryptionKey.getBytes(CHARSET),
encryptKeyAlgorithm);
        cipher.init(1, secretKey);
        byte[] bytes = dataStr.getBytes(CHARSET);
        dataStr = Base64.getEncoder().encodeToString(cipher.doFinal(bytes));
        log.info("Data encrypted. Encrypted data:" + dataStr);
    } catch (Exception e) {
        log.info ("Encrypt data failed");
        log.error ("Data encryption error:", e);
        result.put("code", "500");
        result.put("message", "Encrypt data failed");
        printResult(result);
        return result;
    }
}
result.put("data", dataStr);
printResult(result);
return result;
}
```

```
private void printResult(JSONObject result) {  
    log.info ("Returned data:" + result.toJSONString ());  
}  
}
```

1.2.3 Verifying Callback URL

If the URL for the enterprise application to receive event push is **https://{app_domain}/callback**, OneAccess will send a verification event to this URL when the administrator saves the callback configuration.

URL

POST https://{app_domain}/callback

Request Header

Authorization: Bearer {access_token}

Request Parameters

Table 1-9 Request parameters

Parameter	Type	Description
nonce	String	Random number, which is used together with timestamp to prevent replay attacks on requests.
timestamp	Integer	Timestamp, which is used together with nonce to prevent replay attacks on requests.
eventType	String	Event type. The value is CHECK_URL here.
data	String	Message body. If encryption is disabled, the random string is sent in plaintext. If encryption is enabled, the random string must be decrypted to reveal the content. After decryption, the random and msg fields will be displayed. The msg field contains the plaintext message content.
signature	String	Message signature. The signature is calculated based on the signature key (signatureSaltValue) provided by the enterprise application, along with the timestamp, nonce from the request, and the encrypted message body.

Response Parameters

Table 1-10 Response parameters

Parameter	Type	Description
code	String	Return code. The value 200 indicates success. For details about error codes, see Common Return Codes .
message	String	Response description.
data	String	- If encryption is disabled, the plaintext random string in the request body is returned. - If encryption is enabled, the encrypted random string in the request body is decrypted and the value of the re-encrypted random string is returned. The content must be decrypted, after which the random and msg fields are generated. The msg field contains the plaintext message content.

Example Request

- Example request with message signature and encryption disabled:

```
{
  "nonce": "bqVHvThFGooCRjSf",
  "timestamp": 1573784783795,
  "eventType": "CHECK_URL",
  "data": "random string",
  "signature": ""
}
```

- Example request with message signature and encryption enabled:

```
{
  "nonce": "jmgjjEAlbrMzWmUw",
  "timestamp": 15093849585,
  "eventType": "CHECK_URL",
  "data": "jRqGWO08Tyuxq+ChqGfK7SiPct6MgcUDvzP5CBYnD30=",
  "signature": "K08yDiTEc094KoccOY+VYLQFxxQ="
}
```

Example Response

Status code: 200

Request successful.

- Response example with message signature and encryption disabled:

```
{
  "code": "200",
  "data": " 2852325935078140700",
  "message": "success"
}
```

- Response example with message signature and encryption enabled:

```
{
  "code": "200",
  "message": "success",
  "data": "u5GkfEdZC0EDvDldLWBK/w=="
}
```

1.3 API

1.3.1 Overview

- Enable the application synchronization before synchronizing organizations. For details, see [Step 7](#).
- The prerequisite for the example messages of service events in this section is that the signature and data encryption functions are enabled for the enterprise application.

1.3.2 Adding an Organization Event

This API is used to synchronize new organizations to the application system.

URL

POST `https://{app_domain}/callback`

Request Header

Authorization: Bearer `{access_token}`

Request Parameters

The following request parameters are subject to the organization attributes configured by enterprises. The administrator can set which attributes to synchronize with the target application by referring to [Step 8](#).

Table 1-11 Request parameters

Parameter	Fixed	Type	Description
code	Yes	String(100)	Organization ID, which is globally unique.
name	Yes	String(40)	Organization name, which must be unique at the current level.
parentId	No	String(50)	Parent organization ID.

Response Parameters

Table 1-12 Response parameter

Parameter	Type	Description
id	String(50)	- Organization ID generated after an organization is created for a downstream enterprise application. The ID is sent back to OneAccess as the unique identifier of the organization. - When an organization is modified or deleted, the ID is passed to the downstream application. The ID must match the one in the downstream application. If they differ, the ID returned by the API will overwrite the previous ID.

Example Request

- Example request with message signature and encryption enabled:

```
{
  "nonce": "AmgjjEAlbrMzWmUw",
  "timestamp": 15093849585,
  "eventType": "CREATE_ORGANIZATION",
  "data": "6lu6gxrHydJIXEWxQhUa3UqsWsDZ5LTAo/xU3zhjq9H3syCuFYDYKg==",
  "signature": "K08yDiTEc094KoccOY+VYLQFxxQ="
}
```

- The decrypted JSON string in the request body follows this format:

```
{
  "code": "1000003",
  "name": "Wuhan branch",
  "parentId": "5b183439-36a8-4d08-94ba-61b3c8d40b66"
}
```

Example Response

Status code: 200

Request successful.

- Response example with message signature and encryption enabled:

```
{
  "code": "200",
  "message": "success",
  "data": "j3rRBbc1Q1z1lZM0DDcUGFyaazO3NgnMbgK6UeWT35Druf5zyXg="
}
```

- The decrypted JSON string in the response body follows this format:

```
{
  "id": "6c5bb468-14b2-4183-baf2-06d523e03bd3"
}
```

1.3.3 Modifying an Organization Event

This API is used to synchronize updated organizations to the application system.

URL

POST `https://{app_domain}/callback`

Request Header

Authorization: Bearer `{access_token}`

Request Parameters

Table 1-13 Request parameters

Parameter	Fixed	Type	Description
id	Yes	String(50)	- Organization ID of a downstream enterprise application. - After an organization is successfully synchronized by referring to Adding an Organization Event, an organization is created for a downstream enterprise application. At the same time, an organization ID is generated and sent back to OneAccess as the unique identifier of the organization.
code	Yes	String(100)	Organization ID, which is globally unique.
name	Yes	String(40)	Organization name, which must be unique at the current level.
parentId	No	String(50)	Parent organization ID.

Response Parameters

Table 1-14 Response parameter

Parameter	Type	Description
id	String(50)	- The value is the same as the value of ID in Table 1-13. - Organization ID sent back to OneAccess after the downstream enterprise application updates the organization. - The ID must match the one in the downstream application. If they differ, the ID returned by the API will overwrite the previous ID.

Example Request

- Example request with message signature and encryption enabled:

```
{
  "nonce": "AmgjjEAIbrMzWmUw",
  "timestamp": 15093849585,
  "eventType": "UPDATE_ORGANIZATION ",
  "data": "6xrHydJIxewxQhUa3UqsXHWsDZ5LTAo/xU3zhjq9H3syCuFYDYKg==",
  "signature": "K08yDiTEc094KoccOY+VYLQFxxQ="
}
```

- The decrypted JSON string in the request body follows this format:

Update the organization information based on the organization ID in the request and send the updated attributes to the enterprise application.

```
{
  "id": "6c5bb468-14b2-4183-baf2-06d523e03bd3",
  "code": "1000003",
  "name": "Wuhan branch",
  "parentId": "5b183439-36a8-4d08-94ba-61b3c8d40b66"
}
```

Example Response

Status code: 200

Request successful.

- Response example with message signature and encryption enabled:

```
{
  "code": "200",
  "message": "success",
  "data": "T41FtX1Q1z1LZM0DDcUGFyaazO3NgnMbgK6UeWT35Druf5zyXg="
}
```

- The decrypted JSON string in the response body follows this format:

```
{
  "id": "6c5bb468-14b2-4183-baf2-06d523e03bd3"
}
```

1.3.4 Deleting an Organization Event

This API is used to synchronize deleted organizations to the application system.

URL

POST https://{app_domain}/callback

Request Header

Authorization: Bearer {*access_token*}

Request Parameters

Table 1-15 Request parameter

Parameter	Type	Description
id	String	- Organization ID of a downstream enterprise application. - After an organization is successfully synchronized by referring to Adding an Organization Event, an organization is created for a downstream enterprise application. At the same time, an organization ID is generated and sent back to OneAccess as the unique identifier of the organization.

Response Parameters

Table 1-16 Response parameters

Parameter	Type	Description
code	String	Return code. 200 indicates success.
message	Integer	Description of the error cause.

Example Request

- Example request with message signature and encryption enabled:

```
{
  "nonce": "AmgjjEAJbrMzWmUw",
  "timestamp": 15093849585,
  "eventType": "DELETE_ORGANIZATION ",
  "data": "6lrHydJIXEWxQhUa3UqsXHWsDZ5LTAo/xU3zhjq9H3syCuFYDYKg==",
  "signature": "K08yDiTEc094KoccOY+VYLQFxxQ="
}
```

- The decrypted JSON string in the request body follows this format:

```
{
  "id": "6c5bb468-14b2-4183-baf2-06d523e03bd3"
}
```

Example Response

Status code: 200

Request successful.

```
{
  "code": "200",
  "message": "success"
}
```

1.3.5 Adding a User Event

This API is used to synchronize new users to the application system.

URL

POST `https://{app_domain}/callback`

Request Header

Authorization: Bearer `{access_token}`

Request Parameters

The following request parameters are subject to the identity synchronization parameters configured by enterprises. The administrator can set which attributes to synchronize with the target application by referring to [Step 9](#).

Table 1-17 Request parameters

Parameter	Fixed	Type	Description
username	Yes	String(100)	Username.
name	Yes	String(40)	Real name of the user.
organizationId	Yes	String	Organization ID.
password	Yes	String	Password.
disabled	Yes	Boolean	Whether to disable the function. true : disabled; false : enabled.
firstName	No	String(20)	Name.
middleName	No	String(20)	Middle name.
lastName	No	String(20)	Last name.
mobile	No	String	Mobile phone number.
email	No	String	Email address.
extAttr1	No	--	Extended attribute 1, which is an extended user attribute of an enterprise. Set this attribute based on the site requirements.

Parameter	Fixed	Type	Description
extAttr2	No	--	Extended attribute 2, which is an extended user attribute of an enterprise. Set this attribute based on the site requirements.

Response Parameters

Table 1-18 Response parameter

Parameter	Type	Description
id	String(50)	- User ID generated after a user is created for a downstream enterprise application. The ID is sent back to OneAccess as the unique identifier of the user. - When a user is modified or deleted, the ID is passed to the downstream application. The ID must match the one in the downstream application. If they differ, the ID returned by the API will overwrite the previous ID.

Example Request

- Example request with message signature and encryption enabled:

```
{
  "nonce": "AmgjjEAJbrMzWmUw",
  "timestamp": 1509384...,
  "eventType": "CREATE_USER",
  "data": "6lu6gxrdJIXEWxQhUa3UqsXHWsDZ5LTAo/xU3zhjq9H3syCuFYDYKg==",
  "signature": "K08yDiTEc094KoccOY+VYLQFxxQ="
}
```

- The decrypted JSON string in the request body follows this format:

```
{
  "username": "zhangsan",
  "name": "Tom",
  "mobile": "1899876....",
  "email": "zhangsan@test.com",
  "organizationId": "391551e8-160f-4993-8177-e7b9c5f6....",
  "extAttr1": "value",
  "extAttr2": "value"
}
```

Example Response

Status code: 200

Request successful.

- Response example with message signature and encryption enabled:

```
{
  "code": "200",
  "message": "success",
}
```

```
    "data": "P+rXpWetRg7IP0vdhVgkVwSoZBJeQwY2zhROsJq/HJ+q6tp1qhl9L1+c"  
  }
```

- The decrypted JSON string in the response body follows this format:

```
{  
  "id": "c3a26dd3-27a0-4dec-a2ac-ce211e10...."  
}
```

1.3.6 Modifying a User Event

This API is used to synchronize updated users to the application system.

URL

POST https://{app_domain}/callback

Request Header

Authorization: Bearer *{access_token}*

Request Parameters

Table 1-19 Request parameters

Parameter	Fixed	Type	Description
id	Yes	String(50)	User ID of an enterprise application.
username	Yes	String(100)	Username.
name	No	String(40)	Real name of the user.
firstName	No	String(20)	Name.
middleName	No	String(20)	Middle name.
lastName	No	String(20)	Last name.
organizationId	No	String	Organization ID.
mobile	No	String	Mobile phone number.
email	No	String	Email address.
disabled	Yes	Boolean	Whether to disable the function. true : disabled; false : enabled.

Parameter	Fixed	Type	Description
extAttr1	No	--	Extended attribute 1, which is an extended user attribute of an enterprise. Set this attribute based on the site requirements.
extAttr2	No	--	Extended attribute 2, which is an extended user attribute of an enterprise. Set this attribute based on the site requirements.

Response Parameters

Table 1-20 Response parameter

Parameter	Type	Description
id	String(50)	- The value is the same as the value of ID in Table 1-19. - User ID sent back to OneAccess after the downstream enterprise application updates the user. - The ID must match the one in the downstream application. If they differ, the ID returned by the API will overwrite the previous ID.

Example Request

- Example request with message signature and encryption enabled:

```
{
  "nonce": "AmgjjEAlbrMzWmUw",
  "timestamp": 15093849585,
  "eventType": "UPDATE_USER",
  "data": "6lu6gxrHydJIIXEQhUa3UqsXHWsDZ5LTAo/xU3zhjq9H3syCuFYDYKg==",
  "signature": "K08yDiTEc094KoccOY+VYLQFxxQ="
}
```
- The decrypted JSON string in the request body follows this format:

Update the user information based on the user ID in the request and send the updated attributes to the enterprise application.

```
{
  "id": "c3a26dd3-27a0-4dec-a2ac-ce211e10....",
  "username": "zhangs",
  "name": "Tom 2",
  "mobile": "1867237....",
  "email": "454205....@qq.com",
  "extAttr1": "value",
  "extAttr2": "value"
}
```

Example Response

Status code: 200

Request successful.

- Response example with message signature and encryption enabled:

```
{
  "code": "200",
  "message": "success",
  "data": "P+rXpWetRg7IP0vdhVgkVwSoZBjeQwY2zhROsJq/HJ+q6tp1qhl9L1+c"
}
```

- The decrypted JSON string in the response body follows this format:

```
{
  "id": "c3a26dd3-27a0-4dec-a2ac-ce211e105f97"
}
```

1.3.7 Deleting a User Event

This API is used to synchronize deleted users to the application system.

URL

POST https://{app_domain}/callback

Request Header

Authorization: Bearer {*access_token*}

Request Parameters

Table 1-21 Request parameter

Parameter	Type	Description
id	String	• User ID of a downstream enterprise application. • After a user is successfully synchronized by referring to Adding a User Event, a user is created for a downstream enterprise application. At the same time, a user ID is generated and sent back to OneAccess as the unique identifier of the user.

Response Parameters

Table 1-22 Request parameters

Parameter	Type	Description
code	String	Return code. 200 indicates success.
message	Integer	Description of the error cause.

Example Request

- Example request with message signature and encryption enabled:

```
{
  "nonce": "AmgjjEAJbrMzWmUw",
  "timestamp": "15093849585",
  "eventType": "DELETE_USER",
  "data": "6IXEWxQhUa3UqsXHWsDZ5LTAo/xU3zhjq9H3syCuFYDYKg==",
  "signature": "K08yDiTEc094KoccOY+VYLQFxxQ="
}
```

- The decrypted JSON string in the request body follows this format:

```
{
  "id": "c3a26dd3-27a0-4dec-a2ac-ce211e10...."
}
```

Example Response

Status code: 200

Request successful.

```
{
  "code": "200",
  "message": "success"
}
```

1.4 Common Return Codes

The event callback API uses HTTP status codes to indicate whether the operation is successful or failed. The callback service of the enterprise application returns the following error codes and messages. OneAccess saves these error codes and messages in the application synchronization record.

Table 1-23 Common return codes

Return Code	Description
200	Success.
400	- The <i>XX</i> parameter already exists. For example, the userName parameter already exists. - The <i>XX</i> parameter cannot be empty. For example, the id parameter cannot be empty. - The length of the <i>XX</i> parameter. For example, the name parameter exceeds the specified length. - The format of the <i>XX</i> parameter is incorrect. For example, the format of the email parameter is incorrect.
401	API authentication failed.
404	The <i>XX</i> record does not exist. For example, user not found.
500	System busy. Try again later.

2 Developing Mapping Scripts

OneAccess can map the organization and user attributes of an enterprise to application systems. Application attribute values can be automatically generated using the mapping script. Additionally, the mapped attribute values can be restricted.

The following describes how to develop a mapping definition script.

Code Rule

OneAccess imposes several restrictions on mapping scripts, including disabling Java class, limiting CPU usage time, and restricting memory usage, the script format, and the use of certain functions.

- Do not use Java class.

If the following code is used:

```
var File = Java.type('java.io.File'); File;
```

The following exception will be thrown:

```
java.lang.ClassNotFoundException: java.io.File
```

- Limit the CPU usage time.

By default, the execution time is limited to 1 second. If the execution time exceeds this limit, an exception will be thrown.

If the following code is used:

```
do{}while(true);
```

The following exception will be thrown:

```
ScriptCPUAbuseException
```

- Limit the memory usage.

The default size is 10 MB. If the size exceeds this limit, an exception will be thrown.

If the following code is used:

```
var o={},i=0; while (true) {o[i++] = 'abc'}
```

The following exception will be thrown:

```
ScriptMemoryAbuseException
```

- Restrict the script format.

To ensure proper script formatting, the if, while, and for statements must be enclosed in braces. Failure to do so may result in format errors.

If the following code is used:

```
var o={},i=0; while (true) o[i++] = 'abc';
```

The following exception will be thrown:

```
BracesException
```

- Restrict the use of certain functions.

The following functions cannot be used in the code. If they are included, they will have no effect.

```
print  
echo  
quit  
exit  
readFully  
readLine  
load  
loadWithNewGlobal
```

Example Scripts

- User attributes

The user object can be used in the script and contains all user attributes. The specific attributes are subject to the attribute code in the attribute definition. For details about managing user attributes, see [Managing User Attributes](#). For details about managing account attributes, see [Step 9](#).

Users > User Attributes	
Attribute Name	Code
Username Required	userName
Organization Required	organizationId
Full Name	name
Phone Required	mobile
Email	email
First Name	firstName
Middle Name	middleName
Last Name	lastName
Nick Name	attrNickName
Birthday	attrBirthday
Gender	attrGender
Identity Type	attrIdentityType
Identity Number	attrIdentityNumber
Area	attrArea
City	attrCity

Work	
Attribute Name	Code
Job Number	employeeid
Direct Superior	attrManagerId

- Example 1: Map the user registration time:

```
var createdAt = user.createdAt;
var date = new Date(createdAt);
date.toISOString();
```

- Example 2: Map the mobile phone number of a user and hide the four digits in the middle:

```
var mobile = user.mobile;
var result = "";
if(mobile.length == 15) {
    result = mobile.slice(0,7) + "*****" + mobile.slice(-4);
}
result;
```

- Example 3: Generate a user email address based on the username:

```
var username = user.userName;
username.toLowerCase()+"@huaweicloud.com";
```

- Organization attributes

The organization object can be used in the script and contains all the attributes of the organization.

- Example 1: Map an organization name.

```
var orgName = organization.name;
orgName.toString();
```

- Example 2: Map organization code.

```
var orgCode = organization.code;
orgCode.toString();
```

- Example 3: Map an organization ID.

```
var id= organization.id;
id.toString();
```

- System attributes

Obtain system attributes, such as date.

Example: Map the time for tomorrow:

```
var date = new Date();
date.setDate(date.getDate()+1);
date.toISOString();
```