UserManagement Class

Contains methods to manage end users, for example, to register their verification methods, verify their identity, or remove their personal information.

Namespace

System

Usage

Let users register and deregister identity verification methods. Create custom Login and Verify pages for passwordless login and self-registration. Convert mobile phone numbers to the proper format before registering users. Scramble user data when users request that Salesforce remove their personal information.

This class is available in API version 43.0 and later.

- UserManagement Methods

UserManagement Methods

The following are methods for `UserManagement`.

- **clone()**
  Makes a duplicate copy of the System.UserManagement object.
- **deregisterVerificationMethod(userId, method)**
  Deregisters an identity verification method. Use this method to let users delete an existing verification method.
- **formatPhoneNumber(countryCode, phoneNumber)**
  Formats a mobile phone number for a user. Call this method to ensure that the phone number is formatted properly before updating a user's mobile phone number.
- **initPasswordlessLogin(userId, method)**
  Invokes a verification challenge for passwordless login when creating custom (Visualforce) Login and Verify pages for customers and partners.
- **initRegisterVerificationMethod(method)**
  Invokes a verification challenge for registering identity verification methods with a custom (Visualforce) page. Users can register either their email address or phone number.
- **initSelfRegistration(method, user)**
  Invokes a verification challenge for self-registration when creating a custom (Visualforce) Verify page for Experience Cloud self-registration.
- **initVerificationMethod(method)**
  Initiates a verification service for email, phone (SMS), and the Salesforce Authenticator verification methods.
- **initVerificationMethod(method, actionName, extras)**
  Initiates a verification service for email, phone (SMS), and the Salesforce Authenticator verification methods.
- **obfuscateUser(userId, username)**
  Scrambles users' data on their request when they no longer want their personal data recognized in Salesforce. When you invoke the method for the user, the data becomes

recognized in Salesforce. When you invoke the method for the user, the data becomes anonymous, and you can never recover it.

- **registerVerificationMethod(method, startUrl)**
  Registers an identity verification method. Verification methods can be a time-based one-time password (TOTP), email or text verification code, Salesforce Authenticator, or U2F-compatible security key. End users register verification methods for themselves.
- **sendAsyncEmailConfirmation(userId, emailTemplateId, networkId, startUrl)**
  Send an email message to a user's email address for verification. The message contains a verification link (URL) that the user clicks to verify the email address later on. You can send email verifications in bulk.
- **verifyPasswordlessLogin(userId, method, identifier, code, startUrl)**
  Completes a verification challenge during a passwordless login that uses a custom Verify page (Visualforce only). If the user who is trying to log in enters the verification code successfully, the user is logged in.
- **verifyRegisterVerificationMethod(code, method)**
  Completes registering a user's email address or phone number as a verification method when customizing the identity verification process.
- **verifySelfRegistration(method, identifier, code, startUrl)**
  Completes a verification challenge when creating a custom (Visualforce) Verify page for Experience Cloud site self-registration. If the person who is attempting to register enters the verification code successfully, the user is created and logged in.
- **verifyVerificationMethod(identifier, code, method)**
  Completes the verification service for email, phone (SMS), Salesforce Authenticator, password, or time-based one-time password (TOTP) verification methods.

clone()

Makes a duplicate copy of the System.UserManagement object.

Signature

```
public Object clone()
```

Return Value

Type: User Management

deregisterVerificationMethod(userId, method)

Deregisters an identity verification method. Use this method to let users delete an existing verification method.

Signature

```
public static void deregisterVerificationMethod(Id userId, Auth.VerificationMethod method)
```

Parameters

userId

Type: Id

User ID of the user deregistering the verification method.

method

Type: Auth.VerificationMethod

Verification method used to verify the identity of the user.

Return Value

deregister a phone number when their phone number changes. While only end users can register an identity verification method, you and your users can deregister one. Keep this behavior in mind when you implement a custom registration page.

This method is available in API version 43.0 and later.

> **Note**
>
> This method doesn't support deregistering built-in authenticators.

formatPhoneNumber(countryCode, phoneNumber)

Formats a mobile phone number for a user. Call this method to ensure that the phone number is formatted properly before updating a user's mobile phone number.

Signature

```
global static String formatPhoneNumber(String countryCode, String phoneNumber)
```

Parameters

countryCode

Type: String

A valid country code.

phoneNumber

Type: String

A mobile number that contains from 3 through 49 numeric characters, without the country code. For example, (415) 555-1234.

Return Value

Type: String

Returns a user's mobile phone number in the proper format.

Usage

Use this method to ensure a user's mobile phone number is formatted as required by Salesforce. Then use the method's return value to update the `mobile` field of the user's record. This mobile number is used for SMS-based device activation. For example, mobile phone numbers are stored along with other identity verification methods in Auth.VerificationMethod enum. This method is introduced in API version 43.0. It isn't available in earlier versions.

Here are some acceptable ways that users can enter their mobile number:

- +1, (415) 555-1234 (with plus signs, parentheses, and dashes)
- 1, 4155551234 (only numbers, no symbols)
- 1 , 415-555-1234 (extra spaces)

Now, consider the following examples.

- Correct examples:
  - `formatPhoneNumber('1', '4155551234');`
  - `formatPhoneNumber('+1','(415) 555-1234');`
  - `formatPhoneNumber('1', '415-555-1234');`
- Incorrect example, because the country code and mobile number aren't separated:
  - `formatPhoneNumber(null, '+1 415-555-1234');`

Here's a code example that uses the `formatPhoneNumber` method. It gets the mobile number from the user and converts it to the format required by Salesforce. Then it updates the user's record with the formatted mobile number.

```
global with sharing class PhoneRegistrationController {
    //Input variables
    global String countryCode {get; set;}
    global String phoneNumber {get; set;}

    global String addPhoneNumber()
    {
        if(countryCode == null) return 'Country code is required';
        if(phoneNumber == null) return 'Phone number is required';

        String userId = UserInfo.getUserId();
        User u = [SELECT Id FROM User WHERE Id=:userId LIMIT 1];
        String formatNum = System.UserManagement.formatPhoneNumber(countryCode, phoneNumbe
        u.MobilePhone = formatNum;
        update u;
        return null;
    }

}
```

As long as the country code and phone number are separated, `formatPhoneNumber` returns a value in the proper format.

initPasswordlessLogin(userId, method)

Invokes a verification challenge for passwordless login when creating custom (Visualforce) Login and Verify pages for customers and partners.

Signature

```
public static String initPasswordlessLogin(Id userId, Auth.VerificationMethod method)
```

Parameters

userId

Type: Id

ID of the user who's logging in.

method

Type: Auth.VerificationMethod

Method used to verify the user's identity, which can be EMAIL or SMS.

Return Value

Type: String

Identifier of the verification attempt.

Usage

Use this method along with its paired `verifyPasswordlessLogin` to customize the login experience with your own Visualforce Login and Verify pages. Invoke `initPasswordlessLogin` from the Login page where the user enters an email address or phone number.

> **Note**

First call the `initPasswordlessLogin` method to initiate an authentication challenge. This method:

- Gets the user ID and verification method, such as EMAIL or SMS, from the Login page.
- Looks up the user and checks that the user is unique and active.
- Sends a verification code to the user.
- Adds an entry for the verification attempt to the Identity Verification History log, assigning an identifier to the verification attempt and setting the status to **User challenged, waiting for response**.
- Adds an entry for the Passwordless Login to the Login History log.
- Returns the identifier to `verifyPasswordlessLogin` to link the transactions.

Then call `verifyPasswordlessLogin`, which, if the user enters the verification code correctly, logs in the user.

> **Note**
>
> Users must verify their identity by email address or phone number before they can log in without a password. You can check whether the user is verified from the user's detail page in Setup. Or you can check programmatically with `TwoFactorMethodsInfo`.

initRegisterVerificationMethod(method)

Invokes a verification challenge for registering identity verification methods with a custom (Visualforce) page. Users can register either their email address or phone number.

Signature

```
public static String initRegisterVerificationMethod(Auth.VerificationMethod method)
```

Parameters

method

Type: Auth.VerificationMethod

Method used to verify the user's identity, which can be EMAIL or SMS.

Return Value

Type: String

The method returns an error message if the phone number is already registered, the user isn't a customer or partner, or if the context isn't an Experience Cloud site.

Usage

Use this method along with its paired `verifyRegisterVerificationMethod` to customize the process for registering a user's verification method using a Visualforce Verify page.

First call the `initRegisterVerificationMethod` method to get the verification code sent to the user as input, and validate it. If the verification code isn't valid, it returns an error message.

Example

Here's a code example that registers a user's phone number as a verification method. When the user enters a verification code on the Visualforce page, it invokes `registerUser()`. The method gets the User ID of the user who's registering the verification method and the user's phone number. It also gets the user's registration status to check whether the phone number is verified already. If the user is registered with a different phone number, the number is updated.

```
                User u = [Select MobilePhone, Id from User Where Id=:userId];
                 currPhone = u.MobilePhone;
                 mobilePhone = getFormattedSms(mobilePhone);
               if (mobilePhone != null && mobilePhone != '') {
               u.MobilePhone = mobilePhone;
               update u;
                // We're updating the email and phone number before verifying. Roll back
               //  the change in the verify API if it is unsuccessful.
                exceptionText = System.
                UserManagement.initRegisterVerificationMethod(Auth.VerificationMethod.SMS);
                if(exceptionText!= null && exceptionText!=''){
                    isInit = false;
                    showInitException = true;
                } else {
                     isInit = false;
                     isVerify = true;
                }
                } else {
                    showInitException = true;
                }
            } catch (Exception e) {
                exceptionText = e.getMessage();
                isInit = false;
                showInitException = true;
            }
        }

    public void verifyUser() {
        // Take the user's input for the code sent to their phone number
        exceptionText = System.UserManagement.         verifyRegisterVerificationMethod(code,
        if(exceptionText != null && exceptionText !=''){
        showInitException = true;
        } else {
                //Success
        }
    }
```

initSelfRegistration(method, user)

Invokes a verification challenge for self-registration when creating a custom (Visualforce) Verify page for Experience Cloud self-registration.

Signature

```
public static String initSelfRegistration(Auth.VerificationMethod method, User user)
```

Parameters

method

Type: Auth.VerificationMethod

Method used to verify the identity of the user, which can be EMAIL or SMS.

user

Type: User

User object to insert after successful registration.

Return Value

Type: String

Identifier of the registration attempt.

Usage

Call this method to initiate the authentication challenge, and include a User object to insert if the registration is successful. The method returns the identifier for the self-registration attempt.

> **Note**
>
> If you specify a language in the `LanguageLocaleKey` field on the User object, Salesforce uses this language for verification email and SMS messages.

Then call verifySelfRegistration, which, if the user enters the verification code correctly, logs in the user.

Example

This code contains the result of a verification challenge that registers a new user.

```
String id = System.UserManagement.initSelfRegistration
          (Auth.VerificationMethod.SMS, user);
        Auth.VerificationResult res = System.UserManagement.verifySelfRegistration
          (Auth.VerificationMethod.SMS, id, '123456', null);
        if(res.success == true){
     //redirect
}
```

initVerificationMethod(method)

Initiates a verification service for email, phone (SMS), and the Salesforce Authenticator verification methods.

Signature

```
public static String initVerificationMethod(Auth.VerificationMethod method)
```

Parameters

method

Type: Auth.VerificationMethod

Method used to initiate a verification service for `EMAIL`, `SMS`, or `SALESFORCE_AUTHENTICATOR` verification methods.

Return Value

Type: String

The returned identifier must be passed into `verifyVerificationMethod`.

Usage

Use this method along with its paired `verifyVerificationMethod` to customize a verification service for `EMAIL`, `SMS`, or `SALESFORCE_AUTHENTICATOR` verification methods. The returned identifier from `initVerificationMethod` must be passed into `verifyVerificationMethod`.

First invoke the `initVerificationMethod` method to send a verification code to the user's email or phone number, or to send a push notification to the Salesforce Authenticator. The user then enters the code or approves the push notification. If the verification code isn't valid or the push notification isn't approved, the service returns an error message.

Email Example

This example shows multi-factor authentication using email.

```
    }

    public Auth.VerificationResult verifyVerification() {
    // requiring identifier from the initVerification
    // the code will need to be entered in this method
    return UserManagement.verifyVerificationMethod(identifier, code , Auth.VerificationMethod.
    }
```

initVerificationMethod(method, actionName, extras)

Initiates a verification service for email, phone (SMS), and the Salesforce Authenticator verification methods.

Signature

```
public static String initVerificationMethod(Auth.VerificationMethod method, String actionName,
Map<String,String> extras)
```

Parameters

method

Type: Auth.VerificationMethod

Method used to initiate a verification service for `EMAIL`, `SMS`, or `SALESFORCE_AUTHENTICATOR` verification methods.

actionName

Type: String

For the `SALESFORCE_AUTHENTICATOR` verification method only, the name of the action to display on the Salesforce Authenticator, such as `Connect to My Salesforce Org`. The default action name is `Apex-Defined Activity`.

extras

Type: Map<String,String>

For the `SALESFORCE_AUTHENTICATOR` verification method only, the following extra settings.

- `secure_device_required` –If set to `true`, the user's device must be secured. For example, the user must enter the device's passcode to approve the request. Default setting is `false`.
- `challenge_required` –If set to `true`, the user must complete a biometric challenge, such as face recognition, on the device to approve the request. Default setting is `false`.

Return Value

Type: String

The returned identifier must be passed into `verifyVerificationMethod` method.

Usage

Use this method along with its paired `verifyVerificationMethod` to customize a verification service for `EMAIL`, `SMS`, or `SALESFORCE_AUTHENTICATOR` verification methods. The returned identifier from `initVerificationMethod` must be passed into `verifyVerificationMethod` method.

First invoke the `initVerificationMethod` method to send a verification code to the user's email or phone number, or to send a push notification to the Salesforce Authenticator. The user then enters the code or approves the push notification. If the verification code isn't valid or the push notification isn't approved, the service returns an error message.

Salesforce Authenticator Example

```
public void initVerification() {
// user will receive push notification on their registered MFA devices
identifier = UserManagement.initVerificationMethod(Auth.VerificationMethod.SALESFORCE_AUTH
}

public Auth.VerificationResult verifyVerification() {
// requiring identifier from the initVerification
// user will need to take the action on their registered MFA devices
return UserManagement.verifyVerificationMethod(identifier, '' , Auth.VerificationMethod.SA
}
```

This example shows multi-factor authentication using Salesforce Authenticator. In this example, the *actionName* parameter is set to `Connect to My Salesforce Org` and the `challenge_required` *extra* parameter is set to `true`.

```
public void initVerification() {
Map<String,String> extras = new Map<String,String>();
extras.put('challenge_required','true');
// user will receive push notification in their registered MFA devices
identifier = UserManagement.initVerificationMethod(Auth.VerificationMethod.SALESFORCE_AUTH
}

public Auth.VerificationResult verifyVerification() {
// requiring identifier from the initVerification
// user will need to take the action on their registered MFA devices
return UserManagement.verifyVerificationMethod(identifier, '' , Auth.VerificationMethod.SA
}
```

obfuscateUser(userId, username)

Scrambles users' data on their request when they no longer want their personal data recognized in Salesforce. When you invoke the method for the user, the data becomes anonymous, and you can never recover it. Use this method to set the username to a specific value after it's scrambled.

Signature

```
public static void obfuscateUser(Id userId, String username)
```

Parameters

userId

Type: Id

ID of the user whose data this method scrambles.

username

Type: String

The username after the user's data is scrambled. Sets the value of the scrambled username to a specific string.

Return Value

Type: void

Usage

This method is introduced in API version 43.0. It isn't available in earlier versions.

You can use the `obfuscateUser` method to protect the personal information of your org's users. When invoked, Salesforce permanently scrambles the user's object data and replaces it with

> **Note**
>
> Take care when using this method. The users' data becomes anonymous and can never be recovered.

Considerations

- This method requires that the org's User Management setting, **Scramble Specific Users' Data**, is enabled from Setup.
- This method affects the standard fields of the user object–excluding a few fields such as the user ID, timezone, locale, and profile.
- It is recommended that you note the user's ID and other attributes for post processing, such as the email address, if you want to send the user a confirmation.
- This method changes only the user object. The association between the user and other objects is removed, but no other objects are changed. For example, contact, ThirdPartyAccountLink (TPAL), and user password authentication (UPA) objects remain unchanged.

> **Note**
>
> Assure your admins that invoking this method doesn't trigger an email change notification.

This method is part of our effort to protect users' personal data and privacy. For more information on what you can do to actively protect user data, see Data Protection and Privacy in Salesforce Help.

obfuscateUser(userId)

Scrambles users' data on their request when they no longer want their personal data recognized in Salesforce. When you invoke the method for the user, the data becomes anonymous, and you can never recover it.

Signature

```
public static void obfuscateUser(Id userId)
```

Parameters

userId

Type: Id

ID of the user whose data this method scrambles.

Return Value

Type: void

Usage

This method is introduced in API version 43.0. It isn't available in earlier versions.

You can use the `obfuscateUser` method to protect the personal information of your org's users. When invoked, Salesforce permanently scrambles the user's object data and replaces it with random character strings. The user's detail page exists, but the fields contain meaningless strings of characters. Salesforce merely obfuscates (scrambles) personal data because you can't delete a user in Salesforce; you can only disable or deactivate a user. In other words, the user record remains in the database and this method performs a soft delete.

Considerations

- This method requires that the org's User Management setting, **Scramble Specific Users' Data**, is enabled from Setup.
- This method affects the standard fields of the user object–excluding a few fields such as the user ID, timezone, locale, and profile.
- If you want to send the user a confirmation, it's recommended that you note the user's ID and other attributes for post processing, such as the email address.
- This method changes only the user object. The association between the user and other objects is removed, but no other objects are changed. For example, contact, ThirdPartyAccountLink (TPAL), and user password authentication (UPA) objects remain unchanged.

> **Note**
>
> Assure your admins that invoking this method doesn't trigger an email change notification.

This method is part of our effort to protect users' personal data and privacy. For more information on what you can do to actively protect user data, see Data Protection and Privacy in Salesforce Help.

ObfuscateUser Code Example

```
public class UserManagementController{
    public List <User> users {get; set;}

    public UserManagementController()
    {
        Profile p = [select id from profile where name = 'Customer Community User'];

        users = [select username, id from User where profileId=:p.id AND isactive=true];
    }

    //Use method with extreme caution. Data can't be recovered.
    @InvocableMethod(label='User Management' description='Obfuscate User data and more')
    static public void obfuscate(List<User> users)
    {
        String uid = ApexPages.currentPage().getParameters().get('uid');

        if(uid == null)
            return;

        User u = [select contactId from user where id=:uid];

        System.UserManagement.obfuscateUser(uid);
    }
}
```

registerVerificationMethod(method, startUrl)

Registers an identity verification method. Verification methods can be a time-based one-time password (TOTP), email or text verification code, Salesforce Authenticator, or U2F-compatible security key. End users register verification methods for themselves.

Signature

method

Type: Auth.VerificationMethod

Verification method used to verify the identity of the user.

startUrl

Type: String

Path to the page that users see after they log in.

Return Value

Type:System.PageReference

Usage

Use this method to enable users to complete identity verification, such as multi-factor authentication (MFA), or to log in to their Experience Cloud site without a password. Users register these methods to verify their identity when logging in. You create a custom registration page when implementing mobile-centric passwordless logins. See VerifyPasswordlessLogin.

The `PageReference` returned by `registerVerificationMethod` redirects the user to the Salesforce Verify page. If the user enters the correct code, the user is redirected to the Experience Cloud site page specified by the start URL. For example:

```
PageReference pr = System.UserManagement.registerVerificationMethod(Auth.VerificationMetho
PageReference p =  System.UserManagement.deregisterVerificationMethod(userId,Auth.Verifica
```

This method is available in API version 43.0 and later.

> **Note**
>
> As a security measure, when users add or update mobile numbers in their detail page, they must log in again to verify their identity. As a result, unsaved changes in the app are lost. To disable this security measure, contact Salesforce Support.

sendAsyncEmailConfirmation(userId, emailTemplateId, networkId, startUrl)

Send an email message to a user's email address for verification. The message contains a verification link (URL) that the user clicks to verify the email address later on. You can send email verifications in bulk.

Signature

```
public static Boolean sendAsyncEmailConfirmation(String userId, String emailTemplateId, String networkId, String startUrl)
```

Parameters

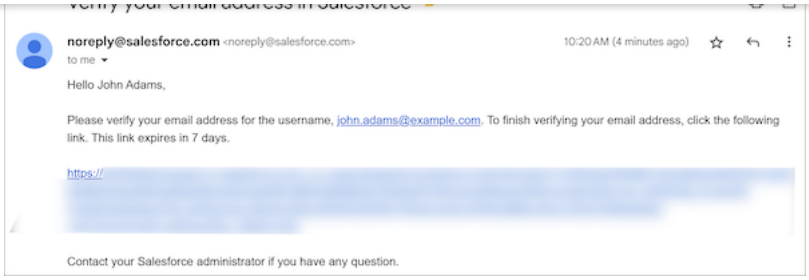
userId

Type: String

ID of the user to receive the email confirmation.

emailTemplateId

Type: String



networkId

Type: String

If verifying email addresses for Experience Cloud site users, the ID of the Experience Cloud site. In addition to external users (such as customers and partners), Experience Cloud site users can include internal users (such as employees) who are members of the site. To verify email addresses for internal users only, set this parameter to `null`.

startUrl

Type: String

The user is redirected to this page after verification, with a success or error message as the parameter. If null, the user is redirected to the login page.

Return Value

Type: Boolean

Indicates whether sending the email message succeeded or failed.

Usage

Sending an async email message is good practice to ensure that users are registered with a valid email address that they truly own. To determine which users receive an email with the verification link, check whether the User Verified Email field in the User detail page is set to true. You can also get this information from the `TwoFactorMethodsInfo` API.

Send async email verification to customers and partners to verify their email address. These users must verify their email address before they can log in with email OTP (passwordless login).

The error code and description are passed as query parameters so that you can process any errors when building a custom landing page.

Example

```
System.UserManagement.sendAsyncEmailConfirmation('005RM000001a0Ox',
'00XRM000000hxnG','0DBRM000000015i', '/s/contactsupport');
```

verifyPasswordlessLogin(userId, method, identifier, code, startUrl)

Completes a verification challenge during a passwordless login that uses a custom Verify page (Visualforce only). If the user who is trying to log in enters the verification code successfully, the user is logged in.

Signature

```
public static Auth.VerificationResult verifyPasswordlessLogin(Id userId,
Auth.VerificationMethod method, String identifier, String code, String startUrl)
```

Parameters

method

Type: Auth.VerificationMethod

Method used to verify the identity of the user, which can be either EMAIL or SMS.

identifier

Type: String

ID of the verification attempt received from the `initPasswordlessLogin` method.

code

Type: String

Code used to verify the identity of the user.

startUrl

Type: String

The page where the user is directed after successful login.

Return Value

Type: Auth.VerificationResult

Result of the verification challenge, which includes the message displayed, and where the user is directed if they enter the verification code correctly.

Usage

Call this method to complete the passwordless login authentication process. It validates the verification method and verification code. It also checks that the identifier is the same as the one returned by `initPasswordlessLogin`.

Example

For an example, see `Auth.VerificationResult`.

verifyRegisterVerificationMethod(code, method)

Completes registering a user's email address or phone number as a verification method when customizing the identity verification process.

Signature

```
public static String verifyRegisterVerificationMethod(String code, Auth.VerificationMethod method)
```

Parameters

code

Type: String

Code used to verify the identity of the user.

method

Type: Auth.VerificationMethod

Method used to verify the identity of the user, which can be either EMAIL or SMS.

Return Value

Type: String

method. This method checks whether the user entered the correct verification code. If the verification code is correct, the method

- Confirms that the user entered the correct verification code
- From the user's detail page, updates the user's verification method status (sets the verification bit)
- Sends an email to the user confirming that a verification method has been added to their record

If the verification code is incorrect, an error message is returned.

> **Note**
>
> If users want to change their email address after registering one, don't use the `initRegisterVerificationMethod` and `verify RegisterVerificationMethod` methods. To enable automatic identity verification for email address changes, from the Identity Verification Setup page, select the field **Require email confirmations for email address changes (applies to users in Experience Builder sites)**.

Example

Here's a code example that registers a user's phone number as a verification method. When the user enters a verification code on the Visualforce page, it invokes `registerUser()`. The method gets the User ID of the user who's registering the verification method and the user's phone number. It also gets the user's registration status to check whether the phone number is verified already. If the user is registered with a different phone number, the number is updated.

```
public void registerUser() {
        try {
            exceptionText='';
            String userId = UserInfo.getUserId();
            User u = [Select MobilePhone, Id from User Where Id=:userId];
             currPhone = u.MobilePhone;
             mobilePhone = getFormattedSms(mobilePhone);
           if (mobilePhone != null && mobilePhone != '') {
           u.MobilePhone = mobilePhone;
           update u;
            // We're updating the email and phone number before verifying. Roll back
           //  the change in the verify API if it is unsuccessful.
            exceptionText = System.
            UserManagement.initRegisterVerificationMethod(Auth.VerificationMethod.SMS);
            if(exceptionText!= null && exceptionText!=''){
                isInit = false;
                showInitException = true;
            } else {
                 isInit = false;
                 isVerify = true;
            }
            } else {
                showInitException = true;
            }
        } catch (Exception e) {
            exceptionText = e.getMessage();
            isInit = false;
            showInitException = true;
        }
    }

public void verifyUser() {
    // Take the user's input for the code sent to their phone number
    exceptionText = System.UserManagement.         verifyRegisterVerificationMethod(code,
    if(exceptionText != null && exceptionText !=''){
    showInitException = true;
    } else {
```

verifySelfRegistration(method, identifier, code, startUrl)

Completes a verification challenge when creating a custom (Visualforce) Verify page for Experience Cloud site self-registration. If the person who is attempting to register enters the verification code successfully, the user is created and logged in.

Signature

```
public static Auth.VerificationResult verifySelfRegistration(Auth.VerificationMethod method,
String identifier, String code, String startUrl)
```

Parameters

method

Type: Auth.VerificationMethod

Method used to verify the identity of the user, which can be either EMAIL or SMS.

identifier

Type: String

The unique identifier received from the `initSelfRegistration` method.

code

Type: String

Code used to verify the identity of the user.

startUrl

Type: String

The page where the user is directed after successful self-registration.

Return Value

Type: Auth.VerificationResult

Result of the verification challenge, which includes the message displayed, and where the user is directed when they enter the verification code correctly.

Usage

By default, when users sign up for your Experience Cloud site with an email address or phone number, Salesforce sends them a verification code and generates a Verify page. This Verify page is where users enter the verification code to confirm their identity. You can replace this Salesforce-generated Verify page with a custom Verify page that you create with Visualforce. Then you invoke the verification process with Apex methods.

First, call the `initSelfRegistration` method, which returns the identifier of the user to create. Then call this `verifySelfRegistration` method to complete the verification process. If the user enters the verification code correctly, the user is created and directed to the page specified in the `startURL`.

This method returns the verification result, which contains the verification status and, if the user is created, the session ID. If the verification method is SMS, the User object must contain a properly formatted mobile number, which is country code, space, and then phone number, for example, `+1 1234567890`. Use `System.UserManagement.formatPhoneNumber` to ensure that the phone number is formatted correctly.

Example

This code contains the result of a verification challenge that registers a new user.

```
        //redirect
    }
```

verifyVerificationMethod(identifier, code, method)

Completes the verification service for email, phone (SMS), Salesforce Authenticator, password, or time-based one-time password (TOTP) verification methods.

Signature

```
public static VerificationResult verifyVerificationMethod(String identifier, String code,
Auth.VerificationMethod method)
```

Parameters

identifier

Type: String

Identifier returned from `initVerificationMethod` for `EMAIL`, `SMS`, and `SALESFORCE_AUTHENTICATOR`.

code

Type: String

Code used to verify the user's identity for `EMAIL`, `SMS`, or `PASSWORD`.

method

Type: Auth.VerificationMethod

Method used to verify the user's identity, which can be `EMAIL`, `PASSWORD`, `SALESFORCE_AUTHENTICATOR`, `SMS`, or `TOTP`.

Return Value

Type: VerificationResult

Usage

Use this method along with its paired `initVerificationMethod` to customize a verification service for `EMAIL`, `SMS`, or `SALESFORCE_AUTHENTICATOR` verification methods. Or use this method alone to provide a complete verification service for `PASSWORD` and `TOTP` verification methods.

This method checks whether the user entered the correct verification code or password. If the verification code or password is correct, the method verifies the user's identity.

If the verification code or password isn't valid, the service returns an error message.

Examples

This example shows multi-factor authentication using email.

```
public void initVerification() {
// user will receive code on their registered verified email
 identifier = UserManagement.initVerificationMethod(Auth.VerificationMethod.EMAIL);
}

public Auth.VerificationResult verifyVerification() {
// requiring identifier from the initVerification
// the code will need to be entered in this method
return UserManagement.verifyVerificationMethod(identifier, code , Auth.VerificationMethod.
}
```

```
public Auth.VerificationResult verifyVerification() {
// user will enter their password as a param in the verifyVerificationMethod for password
return UserManagement.verifyVerificationMethod('', password , Auth.VerificationMethod.PASS
}
```

```
public Auth.VerificationResult verifyVerification() {
// user will enter their registered time-based one-time password (TOTP) code (token)
return UserManagement.verifyVerificationMethod('', code , Auth.VerificationMethod.TOTP);
}
```

DID THIS ARTICLE SOLVE YOUR ISSUE?
Let us know so we can improve!

Share your feedback

© Copyright 2025 Salesforce, Inc. All rights reserved. Various trademarks held by their respective owners. Salesforce, Inc. Salesforce Tower, 415 Mission Street, 3rd Floor, San Francisco, CA 94105, United States

Privacy Information Terms of Service Legal Use of Cookies Trust Cookie Preferences

Your Privacy Choices Responsible Disclosure Contact