ConfigurableSelfRegHandler Interface

Gives you more control over how customers or partners self-register for your Experience Cloud by creating a class that implements `Auth.ConfigurableSelfRegHandler`. You choose the user information to collect, and how users identify themselves–with their email address, phone nu or another identifier. When verified, you create a customer or partner user and log in the user your Experience Cloud site.

Namespace

Auth

Usage

You set up site self-registration declaratively on the Login & Registration (L&R) page of the Administration workspace. When combined with a configurable self-registration setup, the ha class can programmatically fill in user fields, including custom fields, and determine how to cr user and log them in.

When you select the Configurable Self-Reg Page registration page, you choose the user fields collect from the self-registration form, such as last name, first name, username, nickname, mo or email. You also determine the verification method that the user identifies themselves with, which can be email, mobile, or neither. Salesforce generates the `Auth.ConfigurableSelfRegHand` handler, which contains logic on how to create an Experience Cloud site member. Modify the handler to change how users are created, and how collected user information is used.

You can add custom logic to ensure that the email or phone number is unique to the custome partner who's registering. For example, you can add a custom unique field, and write a copy o email or phone number to it. You can also change how the user is created. By default, the use created as a contact associated with the account that you select on the L&R page.

The generated `ConfigurableSelfRegHandler` is located on the Setup Apex Classes page, and be with `AutocreatedConfigSelfReg`, for example, `AutocreatedConfigSelfReg1532475901849`.

For an example, see ConfigurableSelfRegHandler Example Implementation. For more details, Salesforce Customer Identity in *Salesforce Help*.

- **ConfigurableSelfRegHandler Method**
- **ConfigurableSelfRegHandler Example Implementation**
  This Apex code implements the `Auth.ConfigurableSelfRegHandler` interface. After the customer or partner fills out the sign-up page and submits it, the handler is invoked to create an Experience Cloud member with the supplied information. If the registration process requires email or phone verification, the verification process finishes before the `Auth.ConfigurableSelfRegHandler.createUser` is invoked. If verification isn't required, `createUser` is invoked when the customer or partner submits the page.

ConfigurableSelfRegHandler Method

The following is the method for `ConfigurableSelfRegHandler`.

- **createUser(accountId, profileId, registrationAttributes, password)**
  Create a community member from the information that the visitor provided on your community's self-registration page.

Signature

```
public Id createUser(Id accountId, Id profileId, Map<Schema.SObjectField,String>
registrationAttributes, String password)
```

Parameters

accountId

Type: Id

Default account with which the new user is associated. This value comes from the Account f setting on Login and Registration (L&R) page under Registration Page Configuration.

profileID

Type: Id

Profile to assign the new user. This value comes from the Profile field setting on the L&R pag under Registration Page Configuration.

registrationAttributes

Type: Map<Schema.sObjectField,String>

A map of attributes that the registering user entered on the self-registration page. The fields appear on the self-registration page come from the User Fields selected on the L&R page wh the registration type is Configurable Self-Reg Page.

password

Type: String

The password entered by the user if “Include Password” is selected on the L&R page. (If a password isn’t entered, the handler must generate one because a password is required to cr user.)

Return Value

Type: Id

Returns an identifier for the created User object. `Auth.ConfigurableSelfRegHandler` inserts a us and then returns the ID of that user.

ConfigurableSelfRegHandler Example Implementation

This Apex code implements the `Auth.ConfigurableSelfRegHandler` interface. After the custome partner fills out the sign-up page and submits it, the handler is invoked to create an Experienc Cloud member with the supplied information. If the registration process requires email or pho verification, the verification process finishes before the `Auth.ConfigurableSelfRegHandler.creat` is invoked. If verification isn’t required, `createUser` is invoked when the customer or partner submits the page.

Verification occurs by email if the admin chose Email as the verification method when setting the Configurable Self-Reg handler on the Login & Registration (L&R) page. When a visitor click sign-up link from the login page, Salesforce prompts for an email address and then sends a or time password to the specified email address. If the visitor enters the verification code succes on the verify page, the user is created and logged in. Likewise, if the admin chose Text Messag the verification method on the L&R page, the visitor is prompted to enter a phone number. Salesforce sends a challenge (verification code) via SMS to the user. If successful, the user is cr and logged in. Requiring verification before creating a user reduces the number of dummy us cluttering your org.

```
global class AutocreatedConfigSelfReg implements Auth.ConfigurableSelfRegHandler {

    private final Long CURRENT_TIME = Datetime.now().getTime();
    private final String[] UPPERCASE_CHARS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'.split('');
    private final String[] LOWERCASE_CHARS = 'abcdefghijklmnopqrstuvwxyz'.split('');
    private final String[] NUMBER_CHARS = '1234567890'.split('');
    private final String[] SPECIAL_CHARS = '!#$%-_=+<>'.split('');

    // This method is called once after verification (if any was configured).
    // This method should create a user and insert it.
    // Password can be null.
    // Return null or throw an exception to fail creation.
    global Id createUser(Id accountId, Id profileId, Map<SObjectField, String> regis
        User u = new User();
        u.ProfileId = profileId;
        for (SObjectField field : registrationAttributes.keySet()) {
            String value = registrationAttributes.get(field);
            u.put(field, value);
        }

        u = handleUnsetRequiredFields(u);
        generateContact(u, accountId);
        if (String.isBlank(password)) {
            password = generateRandomPassword();
        }
        Site.validatePassword(u, password, password);
        if (u.contactId == null) {
            return Site.createExternalUser(u, accountId, password);
        }
        u.languagelocalekey = UserInfo.getLocale();
        u.localesidkey = UserInfo.getLocale();
        u.emailEncodingKey = 'UTF-8';
        u.timeZoneSidKey = UserInfo.getTimezone().getID();
        insert u;
        System.setPassword(u.Id, password);
        return u.id;
    }
    // Method to autogenerate a password if one isn't passed in.
    // By setting a password for a user, we won't send a
    // welcome email to set the password.
    private String generateRandomPassword() {
        String[] characters = new List<String>(UPPERCASE_CHARS);
        characters.addAll(LOWERCASE_CHARS);
        characters.addAll(NUMBER_CHARS);
        characters.addAll(SPECIAL_CHARS);
        String newPassword = '';
        Boolean needsUpper = true, needsLower = true, needsNumber = true, needsSpeci
        while (newPassword.length() < 50) {
            Integer randomInt = generateRandomInt(characters.size());
            String c = characters[randomInt];
            if (needsUpper && c.isAllUpperCase()) {
                needsUpper = false;
            } else if (needsLower && c.isAllLowerCase()) {
                needsLower = false;
            } else if (needsNumber && c.isNumeric()) {
                needsNumber = false;
            } else if (needsSpecial && !c.isAlphanumeric()) {
                needsSpecial = false;
            }
            newPassword += c;
        }
        newPassword = addMissingPasswordRequirements(newPassword, needsLower, needsU
        return newPassword;
    }

    private String addMissingPasswordRequirements(String password, Boolean addLowerC
        if (addLowerCase) {
            password += LOWERCASE_CHARS[generateRandomInt(LOWERCASE_CHARS.size())];
        }
        if (addUpperCase) {
```

```
            password += SPECIAL_CHARS[generateRandomInt(SPECIAL_CHARS.size())];
        }
        return password;
    }
    // Generates a random number from 0 up to, but not including, max.
    private Integer generateRandomInt(Integer max) {
        return Math.mod(Math.abs(Crypto.getRandomInteger()), max);
    }

    // Loops over required fields that were not passed in to
    // set to some default value.
    private User handleUnsetRequiredFields(User u) {
        if (String.isBlank(u.LastName)){
            u.LastName = generateLastName();
        }
        if (String.isBlank(u.Username)) {
            u.Username = generateUsername();
        }
        if (String.isBlank(u.Email)) {
            u.Email = generateEmail();
        }
        if (String.isBlank(u.Alias)) {
            u.Alias = generateAlias();
        }
        if (String.isBlank(u.CommunityNickname)) {
            u.CommunityNickname = generateCommunityNickname();
        }
        return u;
    }
    // Method to construct a contact for a user.
    private void generateContact(User u, Id accountId) {
        // Add logic here if you want to build your own
        // contact for the use.
    }
    // Default implementation to try to provide uniqueness.
    private String generateAlias() {
        String timeString = String.valueOf(CURRENT_TIME);
        return timeString.substring(timeString.length() - 8);
    }
    // Default implementation to try to provide uniqueness.
    private String generateLastName() {
        return 'ExternalUser' + CURRENT_TIME;
    }
    // Default implementation to try to provide uniqueness.
    private String generateUsername() {
        return 'externaluser' + CURRENT_TIME + '@company.com';
    }
    // Default implementation to try to provide uniqueness.
    private String generateEmail() {
        return 'externaluser' + CURRENT_TIME + '@company.com';
    }
    // Default implementation to try to provide uniqueness.
    private String generateCommunityNickname() {
        return 'ExternalUser' + CURRENT_TIME;
    }
}
```