HeadlessSelfRegistrationHandler Interface

Creates customer and partner users during the Headless Registration Flow.

Namespace

Auth

Usage

The Headless Registration Flow allows you to control user registration experience in a thirc app while using Salesforce to authenticate users and manage their data access. When you this flow, add users in the class that is implementing the `Auth.HeadlessSelfRegistrationHar` interface. This class runs after the user verifies their identity. For a detailed explanation of l registration, see Headless Registration Flow for Private Clients or Headless Registration Flo Public Clients, depending on your app type.

- **HeadlessSelfRegistrationHandler Methods**
  The following are methods for `HeadlessSelfRegistrationHandler`.
- **HeadlessSelfRegistrationHandler Example Implementation**
  This example class implements the `Auth.HeadlessSelfRegistrationHandler` interface create a user. It finds or creates an account to store the new user and creates a cont associate with the account. It then creates the user based on information that your sends to Headless Registration API.

HeadlessSelfRegistrationHandler Methods

The following are methods for `HeadlessSelfRegistrationHandler`.

- **createUser(profileId, data, customUserDataMap, experienceId, password)**
  Returns a User object using information submitted by your off-platform app to Hea Registration API. The User object can be a new user that hasn't been inserted in you database, or it can represent an existing user record. If it's a new User object, Salesf inserts the user record for you.

createUser(profileId, data, customUserDataMap, experienc password)

Returns a User object using information submitted by your off-platform app to Headless Registration API. The User object can be a new user that hasn't been inserted in your org's database, or it can represent an existing user record. If it's a new User object, Salesforce in user record for you.

Signature

```
public User createUser(Id profileId, Auth.UserData data, String customUserDataMap, Str
experienceId, String password)
```

Parameters

data

Type: Auth.UserData

A class that stores information about the user, such as their name and locale.

customUserDataMap

Type: String

A string representation of a JSON object containing custom user information passed in c registration. We recommend that you deserialize this string into the equivalent Apex clas structure. Determine what custom information to collect when you build your app's regis experience.

experienceId

Type: String

A custom value that determines what the end user experiences.

password

Type: String

The user password.

Return Value

Type: User

HeadlessSelfRegistrationHandler Example Implementation

This example class implements the `Auth.HeadlessSelfRegistrationHandler` interface to crea It finds or creates an account to store the new user and creates a contact to associate with account. It then creates the user based on information that your client sends to Headless Registration API.

```
global class ExampleHeadlessReg implements Auth.HeadlessSelfRegistrationHandler
    // TO DO: Update this constant with the actual value for your use case
    private static final String CUSTOMER_ACCOUNT = 'My Account';

    /*
    * Retrieve an existing account or create a new one if it doesn't exist
    *
    * @param accountName - The name of the Account to find or create
    * @return Account - The found or newly created Account record
    */
    private Account findOrCreateAccount(String accountName) {
        List<Account> existingAccounts = [SELECT Id FROM Account WHERE Name=:acc

        if (existingAccounts.isEmpty()) {
            Account newAccount = new Account(Name = accountName);
            insert(newAccount);
            return newAccount;
        }

        return existingAccounts[0];
    }

    /*
    * Create a contact and associate it with an account
    *
    * @param account - The Account object to associate the contact with
    * @param user - The User object containing the first and last name for the c
```

```
        c.lastName = user.lastName;

        insert(c);

        return c;
    }

    //TO DO: Implement any additional password validation that you want in this
    // In this example, the password was already checked to ensure that it compl
    // and the password, if present, is set automatically for the new user when
    private Boolean isPasswordValid(String password) {
        return true;
    }


    global User createUser(Id profileId, Auth.UserData data, String customUserDa
        if (!isPasswordValid(password)) {
            return null;
        }

        User u = new User();
        u.Username = data.username;
        u.ProfileId = profileId;
        u.Email = data.email;
        u.LastName = data.lastName;
        u.FirstName = data.firstName;
        String alias = data.username;
        // Alias must be 8 characters or less
        if (alias.length() > 8) {
            alias = alias.substring(0, 8);
        }
        u.Alias = alias;
        Account a = findOrCreateAccount(CUSTOMER_ACCOUNT);
        Contact c = createContact(a, u);
        u.ContactId = c.Id;
        u.LanguageLocaleKey = UserInfo.getLocale();
        u.LocaleSidKey = UserInfo.getLocale();
        u.EmailEncodingKey = 'UTF-8';
        u.TimeZoneSidKey = UserInfo.getTimezone().getID();

        return u;
    }
}
```

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

Share your fe

DEVELOPER CENTERS
Heroku
MuleSoft
Tableau
Commerce Cloud
Lightning Design System

POPULAR RESOURCES
Documentation
Component Library
APIs
Trailhead
Sample Apps

COMMUN
Trailblazer
Events and
Partner Cor
Blog
Salesforce