|  |  |
| --- | --- |
|  | Azure Active Directory B2C  Module 6 Lab – Work with Tokens, Attributes and Claims |
|  |  |
|  |  |

Contents

[Overview 3](#_Toc458772477)

[Learning Objectives 3](#_Toc458772478)

[1. Collect a new attribute from the user during sign up and send it in the token 3](#_Toc458772479)

[2. Persist an attribute in the directory, and enable user to edit it 4](#_Toc458772480)

[3. Apply a claims transformation to create a new claim 6](#_Toc458772481)

# Overview

During this lab, you will collect a new attribute, include it in a token, make an attribute editable, and apply a claims transformation to create a new claim.

**Estimated time to complete this lab: 45 minutes**.

# Learning Objectives

After completing the exercises in this lab, you will be able to:

* Create an attribute, store it in the schema and use it in an application
* Understand how claims transformations are declared in the policy, the role of InputClaims, OutputClaims and InputParameters
* Use claims transformations to create new claims

## Collect a new attribute from the user during sign up and send it in the token

|  |  |
| --- | --- |
| Task | Detailed Steps |
| Add a claim type for the new attribute in the claims schema section | 1. Open **<tenant.onmicrosoft.com>\_base.xml** from the Advanced Policies Starter Pack and go to the **ClaimsSchema** section 2. Copy the **givenName** claim type (towards at the end of the section) – this has all the important attributes required of a new claim type 3. Paste the claim type so as to create a new one, and update the Id to **eyeColor** (or some other unique value, so it can be referenced from other places in the policy) 4. Update the other elements as necessary replacing **given\_name** with **eyeColor** (or whatever) 5. You could also add a restriction to show how regular expressions can be used to control input as follows:   <ClaimType Id="eyeColor">  <DisplayName>Eye Color</DisplayName>  <DataType>string</DataType>  <DefaultPartnerClaimTypes>  <Protocol Name="OAuth2" PartnerClaimType="eyeColor" />  <Protocol Name="OpenIdConnect" PartnerClaimType="eyeColor" />  <Protocol Name="SAML2" PartnerClaimType="<http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname>" />  </DefaultPartnerClaimTypes>  <UserHelpText>Your eye color.</UserHelpText>  <UserInputType>TextBox</UserInputType>  <Restriction>  <Pattern RegularExpression="^(Amber|Blue|Brown|Gray|Green|Hazel|Other)\*$" HelpText="Must be one of the standard colors, or other" />  </Restriction>  </ClaimType>  **Note:** The user will be able to see the DisplayName and UserHelpText tags. |
| Add the attribute in the self-asserted provider so that you can collect it from the user during local account creation | 1. Find the **<UserJourney Id=”SignUp”** tag 2. Search for **OrchestrationStep Order=”2”** and locate the **TechnicalProfileReferenceId** for the claims exchange **SignUpWithLogonEmailExchange** - it should be **LocalAccountSignUpWithLogonEmail** 3. Search the document for the **TechnicalProfile** whose id is **LocalAccountSignUpWithLogonEmail** and then its **OutputClaims** section 4. Add a claim **<OutputClaim ClaimTypeReferenceId=”eyeColor”** />   **Note:** Essentially an OutputClaim in the SelfAssertedAttributeProvider indicates that this claim needs to be sent back by the provider and thus will be sourced from the user. |
| Include the attribute in the token for the application | 1. Open the **signup** policy (xml file) and add the new **OutputClaim** here as well (<OutputClaim ClaimTypeReferenceId=”eyeColor” />)   **Note:** This claim appears in the RelyingParty section of the policy. This element determines the interaction between AAD B2C and the application that’s making the request.   1. **Save** and **upload** the edited policies in the Azure Portal as before 2. Click the **SignUp** policy on the **All Policies** blade and click **Run Now** 3. Sign up as a new user to test that the new attribute is being collected from the user during local account creation, and sent in the token |

## Persist an attribute in the directory, and enable user to edit it

|  |  |
| --- | --- |
| Task | Detailed Steps |
| Create a new attribute in the B2C admin portal | 1. In **http://portal.azure.com**, signed in as the tenant admin as usual, from the **Settings** blade, click **User attributes** 2. Click **+Add** and enter the name **eyecolor** (or whatever) 3. Provide a suitable description for the attribute and click **Create** 4. **Close** the User attributes blade |
| Use the attribute in a sign-up policy to create the attribute in directory | **Note:** We will use a workaround here. To actually create the attribute in the directory, we need to use it in a sign up policy – this could be any signup policy that you happen to have.   1. Click **Sign-up policies** to open the policies blade, and select an existing sign-up policy 2. Click **Edit** and then **Sign-up attributes** 3. Select **eyecolor** (or whatever) and click **OK** and **Save** (and this attribute is created in the directory when the policy is saved) |
| Find the name of the extension property from Graph API | 1. In a command prompt, navigate to **c:\b2c\B2CGraphClient\bin\debug** 2. Run the command **B2C Get-B2C-Application**   **Note:** This has returned the b2c-extension-app. AAD B2C creates an application in every tenant for handling all extension properties.   1. Save both the **objectId** and **appId** values in your **b2cdata.txt** file on your desktop 2. Run **B2C Get-Extension-Attribute <insert-objectId-from-above>** (this returns a list ofall extension properties on this application) 3. Find the name of the extension property for eyecolor (like extension\_GUID\_eyecolor), and save this also |
| Configure the policy with the application extensions | 1. In the **<TenantName>.onmicrosoft.com\_base.xml** policy, locate the **AAD-Common** technical profile 2. Add a **Metadata** section with two items, one containing the objectId of the extensions application and the other with its appId (both just recorded above):   <Metadata>  <Item Key="ApplicationObjectId">objectId</Item>  <Item Key="ClientId">appId</Item>  </Metadata>  **Note:** These settings allow B2C to locate the application and use it during claims persistence. |
| Update the AAD technical profile to save the new claim during local account sign up | **Note:** You may remember that the LocalAccountSignUpWithLogonEmail self-asserted profile was used during local account creation. If you take a look at this, you will see that it references the ValidationTechnicalProfile called AAD-UserWriteUsingLogonEmail. ValidationTechnicalProfiles are used for validating user data, which means that they are called to validate the user provided data before the user journey moves forward, and this includes saving them in the directory.   1. Add a PersistedClaim in the **AAD-UserWriteUsingLogonEmail** technical profile which will be something like as follows:   <PersistedClaim ClaimTypeReferenceId="eyeColor" PartnerClaimType="extension\_e4735ee1334d498890bc544ca13d2f93\_eyeColor"/> |
| Update the AAD technical profile used during sign in to read the claim and include it in the token | **Note:** The SignIn user journey uses AAD-UserReadUsingObjectId to read user’s claims for local account sign in (in orchestration step 4).   1. In the **AAD-UserReadUsingObjectId TechnicalProfile** add an OutputClaim as follows (similar to the above persisted claim):   <OutputClaim ClaimTypeReferenceId="eyeColor" PartnerClaimType="extension\_e4735ee1334d498890bc544ca13d2f93\_eyeColor"/>   1. Upload the **SignUp** policy and click **Run now** to test the policy and verify that the claim is in the token |
| Display the claim in the edit profile edit so that the user can update it | **Note:** The EditProfile user journey also uses AAD-UserReadUsingObjectId to read the claims after signing in the user. That profile is already updated to read the new claim. It also uses the SelfAsserted-ProfileUpdate profile for displaying the claims to the user, and then updating them in the directory.   1. Still in the **base** policy, find the **TechnicalProfile** called **SelfAsserted-ProfileUpdate**, and add the new claim type as an **InputClaim** and an **OutputClaim** (PartnerClaimType is not needed here):   <InputClaim ClaimTypeReferenceId="eyeColor"/>  **Note:** The InputClaim ensures that the value read from the directory is sent to the SelfAsserted provider so it can display it to the user. OutputClaim gets that value back so it can be sent in the token.   1. This refers to the ValidationTechnicalProfile **AAD-UserWriteProfileUsingObjectId** – so find that and add this claim as a PersistedClaim (this will ensure that the updated claim is written to the directory) 2. Upload the **base** policy 3. Open the **editprofile** policy and add **eyeColor** as an output claim type so that it appears in the token 4. Save the **editprofile** policy and **upload** it 5. Execute the **EditProfile** policy to confirm that the claim has been added to the directory, can be edited by the user, and is included in the token |

## Apply a claims transformation to create a new claim

|  |  |
| --- | --- |
| Task | Detailed Steps |
| Create the claims transformation | 1. In the **base** policy, find the **ClaimsTransformations** section 2. Add a new **ClaimsTransformation** which uses a predefined TransformationMethod, provides the InputClaims and InputParameters, and returns OutputClaims:   <ClaimsTransformation Id="CreateDisplayNameFromGivenNameAndSurname" TransformationMethod="FormatStringMultipleClaims">  <InputClaims>  <InputClaim ClaimTypeReferenceId="givenName" TransformationClaimType="inputClaim1" />  <InputClaim ClaimTypeReferenceId="surname" TransformationClaimType="inputClaim2" />  </InputClaims>  <InputParameters>  <InputParameter Id="stringFormat" DataType="string" Value="{0} {1}" />  </InputParameters>  <OutputClaims>  <OutputClaim ClaimTypeReferenceId="displayName" TransformationClaimType="outputClaim" />  </OutputClaims>  </ClaimsTransformation>  **Note:** This is just the declaration of the transformation; the Id is how the transformation will be referenced. |
| Include the claims transformation in the technical profile | 1. Find the technical profile called **LocalAccountSignUpWithLogonEmail**, and add the following OutputClaimsTransformations section:   <OutputClaimsTransformations>  <OutputClaimsTransformation ReferenceId="CreateDisplayNameFromGivenNameAndSurname"/>  </OutputClaimsTransformations>  **Note:** Now when the technical profile is invoked and claims are returned from the technical profile, this claims transformation will be applied to create (or overwrite) the displayName claim. In the definition of the displayName claim, the partner claim type is set to “name”, so in the token you will see the displayName claims transformation as “name”.   1. **Save** the policy and **upload** it 2. **Execute** the signup policy again, and the name claim (which is our displayName claim transformation) should appear with the correct in the token |