Microsoft

Azure AD B2C Custom Policies

Bring-your-own-identity and Migrating Users

Microsoft Corporation
Published: September 2018
Version: 0.9 (DRAFT)

Author: Philippe Beraud (Microsoft France)
Contributors/Reviewers: Marcelo di Iorio (Microsoft Spain), Kim Cameron, Brandon Murdoch, Ronny Bjones, Jose Rojas (Microsoft Corporation)

For the latest information on Azure Active Directory, please see
http://azure.microsoft.com/en-us/services/active-directory/

Abstract: Azure AD, the Identity Management as a Service (IDaaS) cloud multi-tenant service with proven ability to handle billions of authentications per day, extends its capabilities to manage consumer identities with a new service for Business-to-Consumer (B2C): Azure AD B2C.

Azure AD B2C is "IDaaS for Customers and Citizens" designed with Azure AD privacy, security, availability, and scalability for customer/citizen identity and access management (CIAM). It's a comprehensive, cloud-based, 100% policy driven solution where declarative policies encode the identity behaviors and experiences as well as the relationships of trust and authority inside a Trust Framework (TF).

Whilst the built-in policies in Azure AD B2C leverage a dedicated TF tailored by Microsoft, i.e. the "Microsoft Basic Trust Framework" in which you can set up for your configuration these predefined policies, the custom policies give you full control, and thus allows you to author and create your own Trust Framework through declarative policies. They thus provide you with all the requirements of an Identity "Hub".

This document is intended for IT professionals, system architects, and developers who are interested in understanding the advanced capabilities Azure AD B2C provides with the custom policies, and more especially in this context how to successfully address the most common advanced scenarios.

Table of Content

Notice

This document covers the custom policies[1] now available for evaluation under public preview for all Azure Active Directory B2C (Azure AD B2C) customers. Custom policies are designed primarily for advanced identity pros/developers who need to address the most complex identity scenarios.

This feature set indeed requires developers to configure the Identity Experience Framework (mostly) directly via XML file editing. This method of configuration is powerful but more complex. Advanced identity pros/developers using the Identity Experience Framework should plan to invest some time completing walk-throughs and reading the online reference documentation beyond this series of documents.

For most scenarios, we recommend that you use Azure AD B2C built-in policies[2]. Built-in policies are easier to set up for your configuration. You can use built-in and custom policies in the same Azure Active Directory B2C tenant.

As of this writing, custom policies are in public preview and may be substantially modified before GA. For information, see RELEASE NOTES FOR AZURE ACTIVE DIRECTORY B2C CUSTOM POLICY PUBLIC PREVIEW[3].

This document will be updated to reflect the changes introduced at GA time for custom policies.

This document reflects current views and assumptions be of the date of development and is subject to change. Actual and future results and trends may differ materially from any forward-looking statements. Microsoft assumes no responsibility for errors or omissions in the materials.

THIS DOCUMENT IS FOR INFORMATIONAL AND TRAINING PURPOSES ONLY AND IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT.

---

[1] AZURE ACTIVE DIRECTORY B2C: CUSTOM POLICIES: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-overview-custom

[2] AZURE ACTIVE DIRECTORY B2C: BUILT-IN POLICIES: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-reference-policies

[3] RELEASE NOTES FOR AZURE ACTIVE DIRECTORY B2C CUSTOM POLICY PUBLIC PREVIEW: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-developer-notes-custom

Introduction

Azure AD B2C is a cloud identity service for your consumer-facing web and mobile applications. Azure AD B2C is designed to solve the identity management challenges that have emerged, as economic and competitive pressures drive commercial enterprises, educational institutions, and government agencies to shift their service delivery channels from face-to-face engagements to online web and mobile applications.

Based on standardized protocols, Azure AD B2C is "IDaaS for Customers and Citizens" designed with Azure AD privacy, security, availability, and scalability for customer/citizen identity and access management (CIAM). The "secret sauce" of Azure AD B2C to achieve the above objectives resides in the 100% policy driven Identity Experience Framework that consume fit to purpose declarative policies.

Many of the most frequently used identity use cases can be addresses using the B2C extension in the Azure portal as the developer control surface. However, there some advanced features only available by writing custom user journeys which must be configured directly into policy XML files and uploaded to the B2C tenant. Access to this incremental feature set is available via the custom policies in Azure AD B2C.

> **Note** For a basic level of proficiency with the policy configuration available directly in the B2C Admin portal, see the introduction video BUSINESS-TO-CONSUMER IDENTITY MANAGEMENT WITH AZURE ACTIVE DIRECTORY B2C[4] where all the relevant B2C Admin **portal** settings are.

Objectives of this document

This fourth document discusses the most common advanced identity use cases. This document is indeed intended as a guidance document for implementing these identity use cases so that you can smoothly and seamlessly instantiate your own specific to purpose tailored identity "Hub" based on the advanced capabilities of Azure AD B2C.

For that purpose, it covers a series of common scenarios and depict how to implement them thanks to custom policies.

As far as the related policy definition is concerned, this series of envisaged scenarios leverages the custom policies of the "Starter Pack", and as such override/extend them in many cases to drastically limit the amount of information to provide to enable the intended scenario.

> **Note** For more information on the "Starter Pack" and how to get started with, see the second document of this series.

Non-objectives of this paper

This series of document is not intended as an overview document for the Azure AD offerings but rather focusses on this Azure AD B2C identity service, and more specifically on the custom policies as per currently available public preview.

---

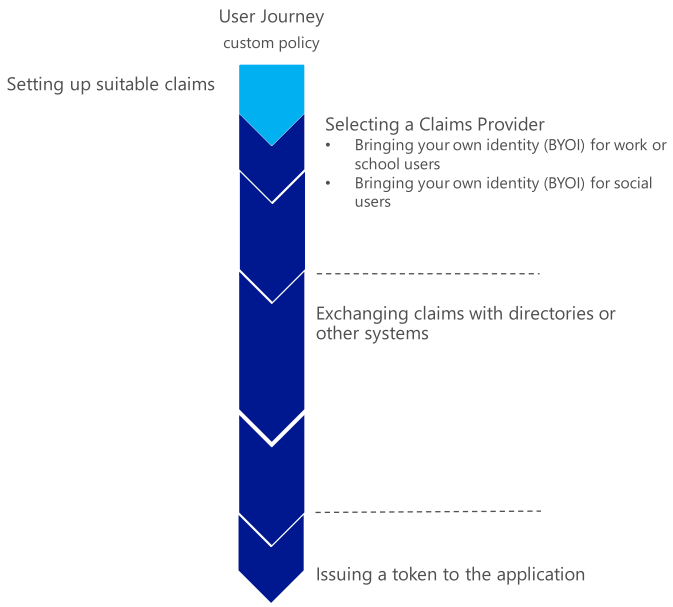
[4] BUSINESS-TO-CONSUMER IDENTITY MANAGEMENT WITH AZURE ACTIVE DIRECTORY B2C: https://channel9.msdn.com/Events/Build/2016/P423

Note For more information, see the article GETTING STARTED WITH AZURE AD[5]. As well as the whitepapers ACTIVE DIRECTORY FROM THE ON-PREMISES TO THE CLOUD[6] and AN OVERVIEW OF AZURE AD[7] as part of the same series of documents.

Organization of this paper

To cover the aforementioned objectives, this document of the series is organized in the following six sections:

- BRINGING YOUR OWN IDENTITY (BYOI) FOR WORK OR SCHOOL USERS.
- BRINGING YOUR OWN IDENTITY (BYOI) FOR SOCIAL USERS.
- EXCHANGING CLAIMS WITH DIRECTORIES OR OTHER SYSTEMS.
- IMPLEMENTING A CUSTOM USER JOURNEY.



- MIGRATING USERS TO YOUR B2C TENANT.
- HELPING TO HANDLE GDPR REQUIREMENTS.

These sections provide the information details necessary to understand the new capabilities introduced by the custom policies in Azure AD B2C that allow successfully implementing the most common use cases with your own (Trust Framework) custom policies.

About the audience

This document is intended for IT professionals, system architects, and developers who are interested in understanding the advanced capabilities Azure AD B2C provides with all the requirements of an Identity

[5] GETTING STARTED WITH AZURE AD: https://docs.microsoft.com/en-us/azure/active-directory/get-started-azure-ad
[6] ACTIVE DIRECTORY FROM THE ON-PREMISES TO THE CLOUD: https://aka.ms/aadpapers
[7] AN OVERVIEW OF AZURE AD: https://aka.ms/aadpapers

"Hub", and in this context how to address the most common use cases based on the already available features as per the currently available public preview.

Bringing your own identity (BYOI) for work or school users

User management and configuring authentication can be a time-consuming process. It can also be error prone if the authentication and identity requirements are complex, leading to possible issues with security. To reduce costs, it is sometimes worthwhile to consider using third party identity providers to manage and authenticate users, although integration multiple IdPs into a solution can also be a challenge. Using Azure AD B2C simplifies many of these tasks.

This section illustrates how to use in a non-exhaustive manner specific products or services to ingrate with Azure AD B2C to "*Bring Your Own Identity*" (BYOI) for work or school users:

- Integrating an Azure AD tenant as a claims provider.
- Integrating AD FS as a claims provider.
- Integrating Salesforce as a claims provider.

This section also covers how to configure password complexity in custom policies for local accounts.

> **Note** In addition to the above use cases, you can also consider the ones in the GitHub repo marcelodiiorio/My-Azure-AD-B2C-use-cases[8], which are found in customers' real situation or requested by peers.

The following subsections depict the related configuration for your B2C tenant and your custom policies based on the "Starter Pack". They assume that you followed the instructions provided in the second document of this series to setup and configure the "Starter Pack".

Integrating an Azure AD tenant as a claims provider

This section shows how to integrate an Azure AD tenant as a claims provider. The litware369.onmicrosoft.com Azure AD tenant will serve as an illustration.

> **Note** For more information, see articles AZURE ACTIVE DIRECTORY B2C: SIGN IN BY USING AZURE AD ACCOUNTS[9] and AZURE ACTIVE DIRECTORY B2C: ALLOW USERS TO SIGN IN TO A MULTI-TENANT AZURE AD IDENTITY PROVIDER USING CUSTOM POLICIES[10].

If you're not interested in this use case, you can skip this entire section and jump to the next one.

To do so, you will have to:

1. Create an Azure AD application for your B2C tenant.
2. Add the Azure AD application key in your B2C tenant.
3. Update the custom policy to configure the organizational Azure AD tenant as a claims provider in the intended user journey(s).

---

[8] marcelodiiorio/My-Azure-AD-B2C-use-cases: https://github.com/marcelodiiorio/My-Azure-AD-B2C-use-cases
[9] AZURE ACTIVE DIRECTORY B2C: SIGN IN BY USING AZURE AD ACCOUNTS: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-setup-aad-custom
[10] AZURE ACTIVE DIRECTORY B2C: ALLOW USERS TO SIGN IN TO A MULTI-TENANT AZURE AD IDENTITY PROVIDER USING CUSTOM POLICIES: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-setup-commonaad-custom

4. Upload the custom policy.
5. Test the custom policy using **Run Now**.

The next sections detail the above.

Creating an Azure AD application

To enable sign-in for users from a specific Azure AD organization, you need to prior register an application within the organizational Azure AD tenant.

We use "fabrikam369.onmicrosoft.com" for the organizational Azure AD tenant and "litware369b2c.onmicrosoft.com" as the B2C tenant in the following instructions.

To illustrate the core principles that pertain to the integration of any OAuth 2.0 based identity provider, we cover hereafter the declaration of your B2C tenant as an application in this social identity provider.

Proceed with the following steps:

1. Open a browsing session and navigate to the Azure portal at https://portal.azure.com.
2. Log in to the Azure portal as an account with administrative privileges in your Azure tenant.
3. On the upper right corner, select your account. From the **Directory** list, choose the organizational Azure AD tenant where you want to register your application, For example **fabrikam369.onmicrosoft.com** in our illustration.
4. Click **Azure Active Directory** on the left navigation menu. (You may need to find it by selecting **More services>**.)
5. Select **App registrations** and click **New application registration**. A new blade opens.

6. Provide the following entries:
    a. In **Name**, enter "*Azure AD B2C (litware369b2c.onmicrosoft.com)*" for the web application.

b. For **Application type**, leave **Web app / API** selected.

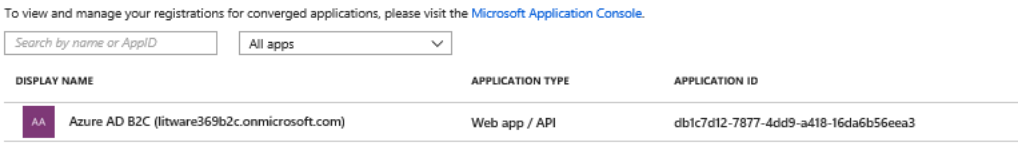
c. In **Sign-on URL**, specify:

https://login.microsoftonline.com/te/<*your_b2c_tenant*>.onmicrosoft.com/oauth2/authresp

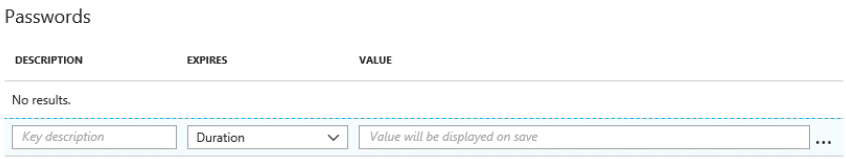
where <*your_b2c_tenant*> by the name of your B2C tenant. (The value for <*your_b2c_tenant*> must be all lowercase.) For example in our configuration:

https://login.microsoftonline.com/te/litware369b2c.onmicrosoft.com/oauth2/authresp

7. Select **Create**.
8. Select **All apps**.



9. Note the **application ID**, here "db1c7d12-7877-4dd9-a418-16da6b56eea3". You will need them later to configure this organizational Azure AD tenant as an identity provider in your B2C tenant. This value will be referred as to the "*YourAppIDValue*" value.
10. Select the newly created web application.
11. Select **Settings** > **Keys**.



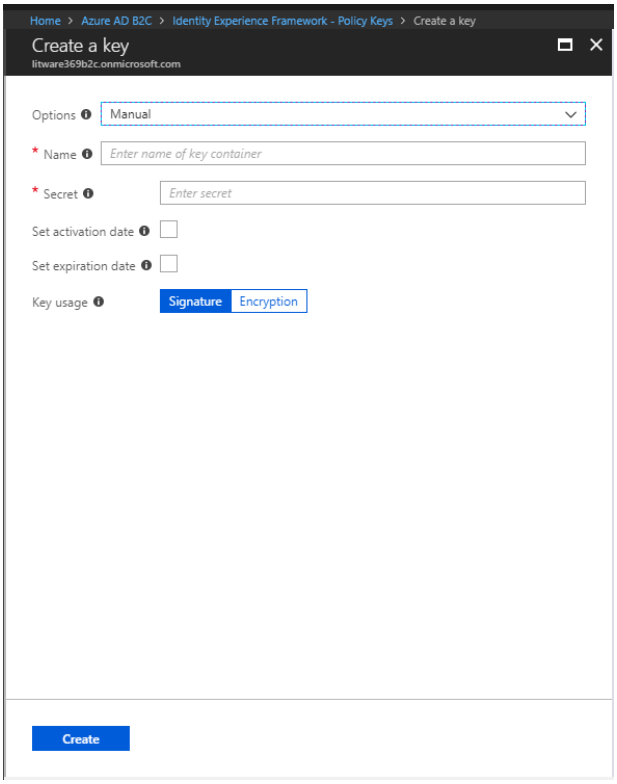
12. Under **Passwords**, enter the key description, select a duration, and then click **Save**. The value of the key is displayed. Copy it because you will use it in the steps in the next section. It will be referred as to the "*YourAppSecretValue*" value.

Adding the Azure AD application key in your B2C tenant

Federation with Azure AD organizational accounts requires a client secret for the organizational Azure AD tenant to trust Azure AD B2C on behalf of the above application. This secret, i.e. the above "*YourAppSecretValue*" value need to be stored in your B2C tenant.

Proceed with the following steps:

1. Log in to the Azure portal as an account with administrative privileges in your Azure tenant, and then switch to your B2C tenant, for example **litware369b2.onmicrosoft.com** in our illustration.
2. Select **All services**, type **Azure AD BC** in the search field, and then select **Azure AD B2C**.
3. Select **Identity Framework Experience - PREVIEW**.
4. Select **Policy Keys** from left menu.
5. Select **+Add**.
6. For **Options**, select **Manual**.



7. For **Key usage**, leave **Signature** selected.
8. In **Name**, choose a name that matches your Azure AD tenant name. For example, "*Fabrikam369Secret*".
9. In **Secret**, enter the Azure AD app secret key you recorded earlier, i.e. the above "*YourAppSecretValue*" value.
10. Select **Create**.

Once the operation complete, a key **B2C_1A_Fabrikam369Secret** should be created and listed.

Updating the custom policy

Defining the organization Azure AD tenant as a claim provider

To sign in by using the Azure AD organizational account, you need to define the Azure AD tenant as a claims provider in your B2C tenant. In other words, you need to specify an endpoint that Azure AD B2C communicates with. The endpoint provides a set of claims that are used by Azure AD B2C to verify that a specific user has authenticated.

It's time to define Azure AD as a claims provider with an associated technical profile. This technical profile is named **Fabrikam369-OIDC** hereafter.

Proceed with the following steps:

1. Navigate to the *SocialAndLocalAccounts* folder in the "Starter Pack".
2. Open the extension policy file, i.e. the *TrustFrameworkExtensions.xml* file, using an XML editor of your choice, for instance Visual Studio (Code).

3. Scroll down to the *ClaimsProviders* section.
4. Add the following *ClaimsProvider* element to the list in the *ClaimsProviders* node.

```
<!-- Organizational Azure AD claims provider -->
<ClaimsProvider>

    <Domain>fabrikam369.onmicrosoft.com</Domain>
    <DisplayName>Login using Fabrikam369</DisplayName>

    <TechnicalProfiles>

        <TechnicalProfile Id="Fabrikam369-OIDC">
            <DisplayName>Fabrikam369 Employee</DisplayName>
            <Description>Login with your Fabrikam369 account</Description>

            <Protocol Name="OpenIdConnect"/>
            <OutputTokenFormat>JWT</OutputTokenFormat>
            <Metadata>

                <Item Key="METADATA">https://login.windows.net/your_AzureAD_tenant/.well-known/openid-configuration</Item>
                <Item Key="ProviderName">https://sts.windows.net/your_AzureAD_tenant_id/</Item>
                <Item Key="client_id">your_AzureAD_client_id</Item>
                <Item Key="IdTokenAudience">your_AzureAD_client_id</Item>

                <Item Key="response_types">id_token</Item>
                <Item Key="UsePolicyInRedirectUri">false</Item>
            </Metadata>
            <CryptographicKeys>
                <Key Id="client_secret" StorageReferenceId="B2C_1A_Fabrikam369Secret"/>
            </CryptographicKeys>
            <OutputClaims>
                <OutputClaim ClaimTypeReferenceId="socialIdpUserId" PartnerClaimType="oid"/>
                <OutputClaim ClaimTypeReferenceId="tenantId" PartnerClaimType="tid"/>
                <OutputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="given_name" />
                <OutputClaim ClaimTypeReferenceId="surName" PartnerClaimType="family_name" />
                <OutputClaim ClaimTypeReferenceId="displayName" PartnerClaimType="name" />
                <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="azureADAuthentication" />
                <OutputClaim ClaimTypeReferenceId="identityProvider" DefaultValue="AzureAD" />
            </OutputClaims>
            <OutputClaimsTransformations>
                <OutputClaimsTransformation ReferenceId="CreateRandomUPNUserName"/>
                <OutputClaimsTransformation ReferenceId="CreateUserPrincipalName"/>
                <OutputClaimsTransformation ReferenceId="CreateAlternativeSecurityId"/>
                <OutputClaimsTransformation ReferenceId="CreateSubjectClaimFromAlternativeSecurityId"/>
            </OutputClaimsTransformations>
            <UseTechnicalProfileForSessionManagement ReferenceId="SM-Noop"/>
        </TechnicalProfile>
    </TechnicalProfiles>
</ClaimsProvider>
```

Note For more information, see section § *Specifying a technical profile for an OpenID Connect claims provider* in the sixth document of this series, along with the implementation notes for the technical profile if any.

5. Under the *ClaimsProvider* node:
    a. Update the value for the *Domain* element to a unique value that both reflect the organizational Azure AD tenant and that can be used to distinguish it from other identity providers.
    b. Update the value for the *DisplayName* element to a friendly name for the claims provider. This value is not currently used.
6. To get a token from the Azure AD endpoint, you need to define the protocols that Azure AD B2C should use to communicate with Azure AD. This is done inside the *TechnicalProfile* element of the *ClaimsProvider* section.
7. Update the ID of the *TechnicalProfile* node. This ID is used to refer to this technical profile from other parts of the policy.

8. Update the value for the *DisplayName* element. To reflect the organizational Azure AD tenant being referenced. This value will be displayed on the sign-in button on your sign-in screen.
9. Likewise, update the value for the *Description* element.
10. You need to update the *Metadata* section in the XML snippet to reflect the configuration settings for your specific organizational Azure AD tenant. In the above XML snippet, update the metadata values as follows:
    a. Set <Item Key="METADATA"> to:

       https://login.windows.net/*your_AzureAD_tenant*/.well-known/openid-configuration

       where *your_AzureAD_tenant* is your Azure AD tenant name, for example fabrikam369.onmicrosoft.com

       https://login.windows.net/fabrikam369.onmicrosoft.com/.well-known/openid-configuration
11. Open a browsing and navigate to the METADATA URL that you just updated.
12. In the browser, look for the 'issuer' object and copy its value. It should look like the following: https://sts.windows.net/*your_AzureAD_tenant_id*/. For example, in our configuration:

    https://sts.windows.net/79943d36-a830-454f-89aa-9c775b83a809/
13. Paste the value for <Item Key="ProviderName"> in the XML snippet.
14. Replace the value your_AzureAD_client_id of <Item Key="client_id"> with the client ID of your Azure AD application, i.e. the above the "*YourAppIDValue*" value.
15. Replace the value your_AzureAD_client_id of <Item Key="IdTokenAudience"> with the client ID of your Azure AD application, i.e. the above the "*YourAppIDValue*" value.

    In addition, federation with Azure AD requires a client secret for Azure AD to trust Azure AD B2C on behalf of the application. The above XML code snippet already references the same Azure AD secret named **B2C_1A_Fabrikam369Secret** that was created through the B2C blade of the Azure portal if you have followed the procedure outlined in the previous section.
16. Save the XML file.

Registering the Azure AD claims provider to the Sign-Up or Sign-In (SUSI) user journey

To register the Azure AD claims provider to the Sign-Up or Sign-In (SUSI) user journey, proceed with the following steps:

1. Still from the *SocialAndLocalAccounts* folder in the "Starter Pack", open the *TrustFrameworkExtensions.xml* file using an XML editor of your choice.
2. Find the *UserJourneys* element. If the element doesn't exist, add one (or uncomment it).
3. Copy the entire content of *UserJourneys* node from the *TrustFrameworkBase.xml* file as a child of the *UserJourneys* element in the *TrustFrameworkExtensions.xml* file. If you already copied this content, you can skip this step.
4. Find the *UserJourney* element with attribute value Id="SignUpOrSignIn".

5. Locate the *OrchestrationStep* element with attribute Order="1" value under this element.
6. Add the following XML snippet under the *ClaimsProviderSelections* node:

```
<ClaimsProviderSelection TargetClaimsExchangeId="Fabrikam369Exchange" />
```

7. Locate the *OrchestrationStep* element with attribute Order="2" in the *UserJourney* node and add the following XML snippet under the *ClaimExchanges* node:

```
<ClaimsExchange Id="Fabrikam369Exchange" TechnicalProfileReferenceId="Fabrikam369-OIDC" />
```

8. Save the XML file.

Uploading the custom policy to your B2C tenant

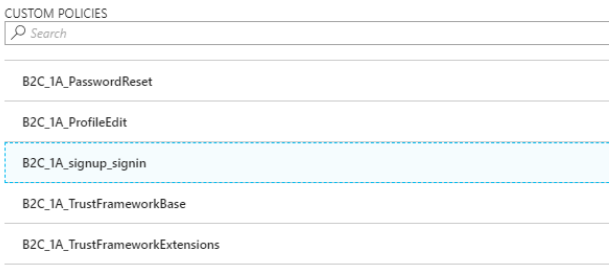
Proceed with the following steps:

1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
2. Select **Upload Policy**.
3. Check **Overwrite the policy if it exists**.
4. In **Upload policy**, select the *TrustFrameworkExtensions.xml* policy file.
5. Click **Upload** and ensure that it does not fail the validation.

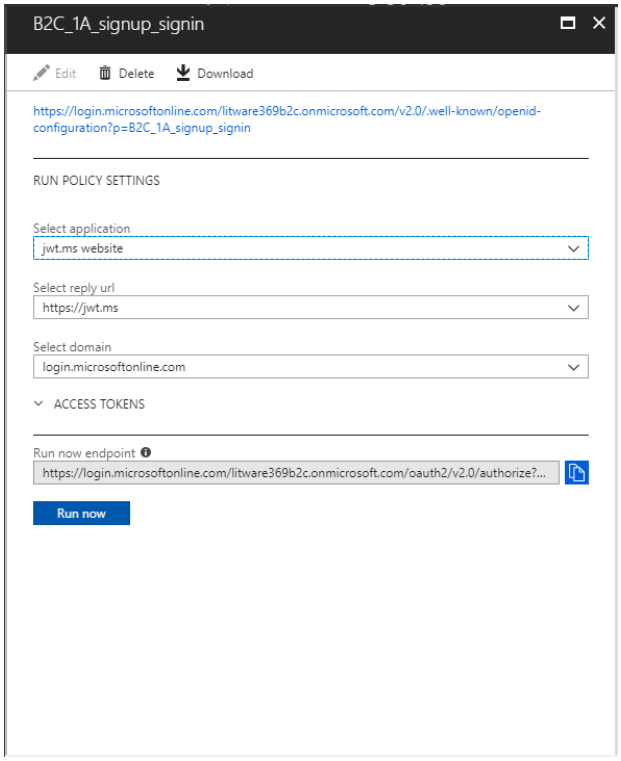
Testing the custom policy by using Run Now

Proceed with the following steps:

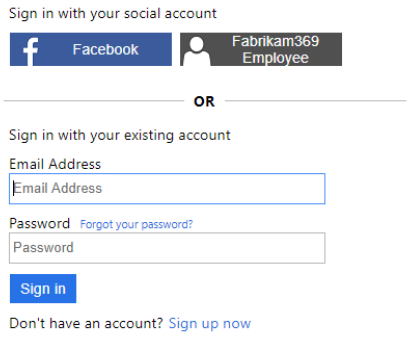
1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.



2. Click the **B2C_1A_signup_signin** custom policy to see its detail view. A new blade opens.



3. Click **Run now**.
4. The sign-in page should include the option to sign in with the organizational Azure AD tenant.



5. You should be able to sign in using your Azure AD account.

Integrating AD FS as a claims provider

This section covers how to integrate an on-premises AD FS as a claims provider in your custom policies.

This can be achieved via both the WS-Fed and the SAML 2.0 standard.

Integrating AD FS as a claim provider via WS-Fed

This section shows you, in this use case, how to configure on-premises AD FS as a claims provider (via the WS-Fed protocol) in your B2C tenant.

As of this writing, the use of the WS-Fed protocol should be considered as under DEVELOPMENT and is NOT part of the features included in the public preview. For information, see RELEASE NOTES FOR AZURE ACTIVE DIRECTORY B2C CUSTOM POLICY PUBLIC PREVIEW[11].

If you're not interested in this use case, you can skip this entire section and jump to the next one.

It notably covers how to:

1. Create a relying party trust in the on-premises AD FS infrastructure.
2. Update the custom policy to configure AD FS as a claims provider in the intended user journey(s).
3. Upload the custom policy.
4. Test the custom policy using **Run Now**.

The next sections detail the above.

Creating a relying party trust in AD FS

In this use case, your B2C tenant acts as a relying party from the AD FS perspective and obtains a security token from the AD FS as a claims provider.

The configuration on the AD FS side is straightforward as one can expect from the solution. This consists as usual in:

1. Creating a relying trust for Azure AD B2C.
2. Configuring the issuance transform rules.

The next sections detail the above. AD FS in Windows Server 2012 R2 will serve hereafter as an illustration for the configuration in AD FS. Differences with AD FS in Windows Server 2016 will be highlighted on purpose if any.

Creating a relying party trust

The **Add-ADFSRelyingPartyTrust** cmdlet enables to add a new relying party trust to your AD FS deployment:

```
Add-ADFSRelyingPartyTrust -Identifier <String[]> -Name <String> -WSFedEndpoint <Uri>
```

Note	For more information, see article ADD-ADFSRELYINGPARTYTRUST[12].

Note	For information on how to create a relying party trust manually, see article CREATE A RELYING PARTY TRUST[13].

As per WS-Fed specification being used in this scenario, the WS-Fed request must include a *wtRealm* parameter, which corresponds in AD FS to the above *Identifier* argument of the cmdlet.

---

[11] RELEASE NOTES FOR AZURE ACTIVE DIRECTORY B2C CUSTOM POLICY PUBLIC PREVIEW: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-developer-notes-custom

[12] ADD-ADFSRELYINGPARTYTRUST: https://docs.microsoft.com/en-us/powershell/module/adfs/add-adfsrelyingpartytrust?view=winserver2012r2-ps

[13] CREATE A RELYING PARTY TRUST: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/dn486828(v=ws.11)

The value of this parameter should be set as follows with Azure AD B2C unless specified otherwise in the metadata of the related WS-Fed claims provider technical profile in the policy XML file:

https://login.microsoftonline.com/te/<*your_b2c_tenant*>/<*your_base_policy*>

Where *your_b2c_tenant* is the B2C tenant, and *your_base_policy* the base policy your custom policy being executed inherits from. For example, in our configuration:

For example:

https://login.microsoftonline.com/te/litware369b2c.onmicrosoft.com/B2C_1A_TrustFrameworkBase

Generally speaking, AD FS when acting as a claims provider must normally be configured to return the issued SAML security token to specific WS-Fed endpoint addresses. Hence, as per WS-Fed specification, the WS-Fed request also includes a *wReply* parameter, which is the URL to which the response token should be posted. This parameter corresponds to the *WSFedEndpoint* argument of the cmdlet.

The *WSFedEndpoint* argument must be set to:

https://login.microsoftonline.com/te/<*your_b2c_tenant*>/<*your_custom_policy*>/WSFED/SSO/ASSERTIONCONSUMER

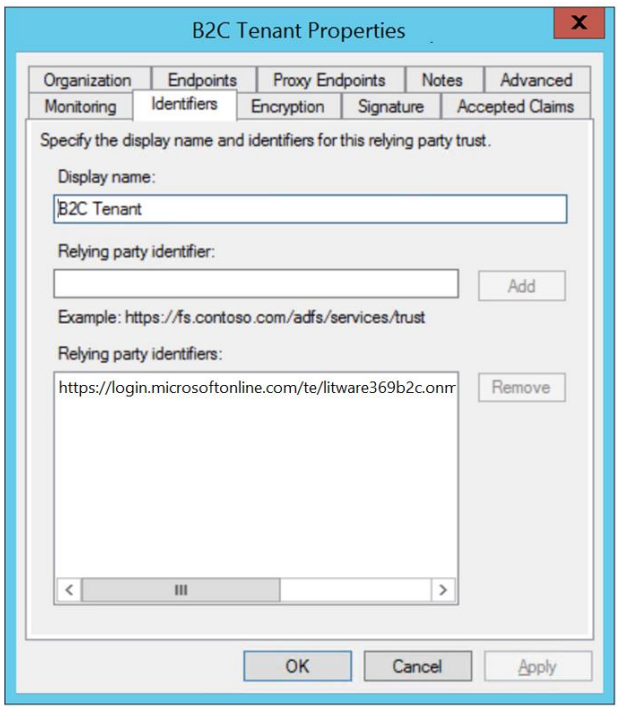
Where *your_b2c_tenant* is the B2C tenant, and *your_custom_policy* the custom policy being executed. For example, in our configuration:

https://login.microsoftonline.com/te/litware369b2c.onmicrosoft.com/B2C_1A_Signup_Signin/WSFED/SSO/ASSERTIONCONSUMER

Considering the above, to create a new relying party trust named **B2C Tenant** to your AD FS, run the following command:

```
PS> $wsRealm = "https://login.microsoftonline.com/te/litware369b2c.onmicrosoft.com/B2C_1A_TrustFrameworkBase"
PS> $wsReply =
"https://login.microsoftonline.com/te/litware369b2c.onmicrosoft.com/B2C_1A_Signup_Signin/WSFED/SSO/ASSERTIONCONSUMER"
PS> Add-ADFSRelyingPartyTrust -Identifier $wsRealm -Name 'B2C Tenant' -WSFedEndpoint $wsReply
```

As per above configuration, the **Identifier** tab of the **B2C Tenant** relying party trust contains the above *wtRealm* value under **Relying party identifiers**.



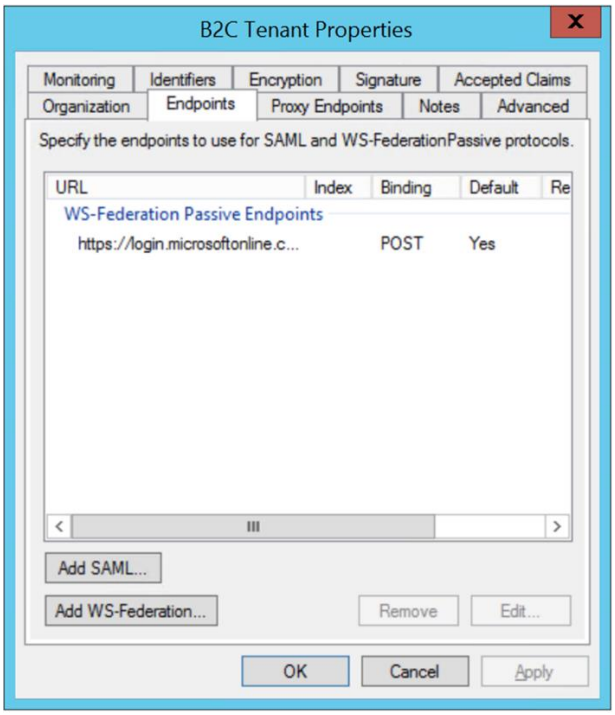
In addition to:

https://login.microsoftonline.com/te/litware369b2c.onmicrosoft.com/B2C_1A_Signup_Signin/WSFED/SSO/ASSERTIONCONSUMER

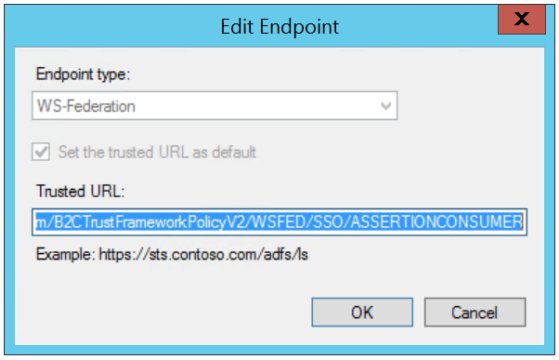
Add the following two Relying Party identifiers:

1. https://login.microsoftonline.com/te/litware369b2c.onmicrosoft.com/B2C_1A_Signup_Signin
2. https://login.microsoftonline.com/te/litware369b2c.onmicrosoft.com/B2C_1A_TrustFrameworkBase

Likewise, the **endpoint** tab contains the above *wReply* value listed under **WS-Federation Passive Endpoints**.



By double-clicking the related line, you can edit the value if needed.



Configuring AD FS issuance transform rules

The previous relying party trust is where the federation trust between AD FS as a claims provider and your B2C tenant as a relying party is configured.

AD FS is a security token service (STS) that relies on a claims-based model. In this model, the claims pipeline represents the path that claims must follow through the service before they can be issued as part of a SAML token.

> **Note** For additional detail on the claims pipeline, see article THE ROLE OF THE CLAIMS PIPELINE[14].

AD FS manages the entire end-to-end process of flowing claims through the various stages of the claims pipeline, which also includes the processing of different claim rule sets by the claim rule-based engine.

---

[14] THE ROLE OF THE CLAIMS PIPELINE: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/ee913585(v=ws.11)

Note For additional detail on the claim rule-based engine, see article THE ROLE OF THE CLAIMS ENGINE[15].

In the considered use case, the claim engine controls which users have access to the Azure AD B2C relying party based on the issuance authorization rules, and then it issues outgoing claims to your B2C tenant based on issuance transform rules.

Consequently, issuance transform rules must be at least configured to define the claims to be returned to Azure AD B2C.

Each of these claims should have an existing claim description in AD FS, which is comparable to the claims schema definition in the XML policy file. The **Add-AdfsClaimDescription** cmdlet enables to add such a definition to AD FS if required.

Note For more information, see article ADD-ADFSCLAIMDESCRIPTION[16].

The **New-ADFSClaimRuleSet** cmdlet allows to create a set of claims rules.

Note For more information, see article NEW-ADFSCLAIMRULESET[17].

Note For information on how to create claims rule for a Relying Party Trust, see article CHECKLIST: CREATING CLAIM RULES FOR A RELYING PARTY TRUST[18].

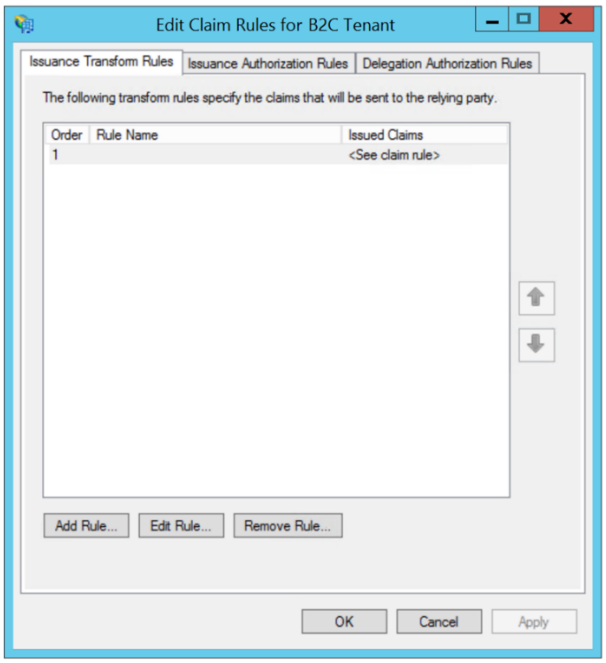
The **Set-ADFSRelyingPartyTrust** cmdlet allows then to add this set of claims rules as issuance transform rules to an existing relying party trust identified via its *wtRealm* value.

Note For more information, see article SET-ADFSRELYINGPARTYTRUST[19].

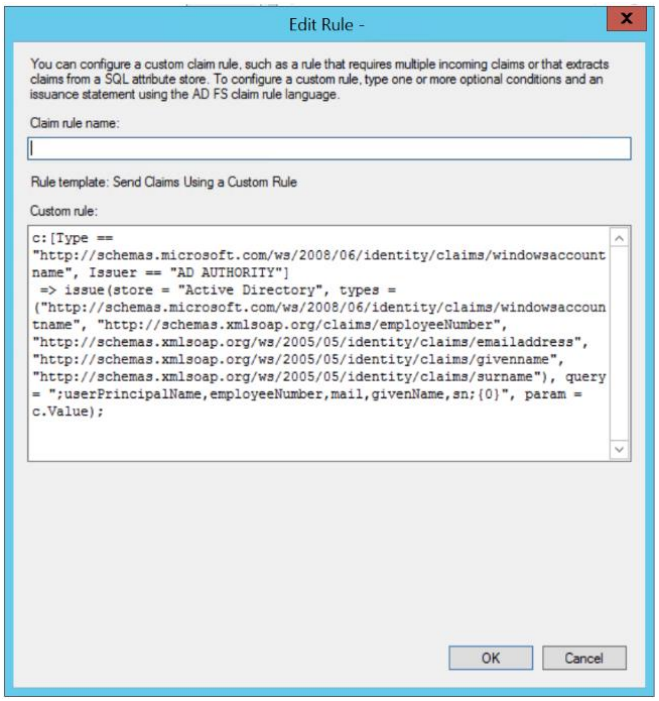
To configure issuance transform rules for the previously created relying party trust named **B2C Tenant**, run the following commands:

```
PS> $wsRealm = "https://login.microsoftonline.com/te/litware369b2c.onmicrosoft.com/B2C_1A_TrustFrameworkBase"
PS> $ruleSet = New-ADFSClaimRuleSet -ClaimRule 'c:[Type ==
"http://schemas.microsoft.com/ws/2008/06/identity/claims/windowsaccountname", Issuer == "AD AUTHORITY"] => issue(store =
"Active Directory", types = ("http://schemas.microsoft.com/ws/2008/06/identity/claims/windowsaccountname",
"http://schemas.xmlsoap.org/claims/employeeNumber",
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress",
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname",
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname"), query =
";userPrincipalName,employeeNumber,mail,givenName,sn;{0}", param = c.Value);'
PS> Set-ADFSRelyingPartyTrust -TargetIdentifier $wsRealm -IssuanceTransformRules $ruleSet.ClaimRulesString
```

[15] THE ROLE OF THE CLAIMS ENGINE: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/ee913582(v=ws.11)
[16] ADD-ADFSCLAIMDESCRIPTION: https://docs.microsoft.com/en-us/powershell/module/adfs/add-adfsclaimdescription?view=winserver2012r2-ps
[17] NEW-ADFSCLAIMRULESET: https://docs.microsoft.com/en-us/powershell/module/adfs/new-adfsclaimruleset?view=winserver2012r2-ps
[18] CHECKLIST: CREATING CLAIM RULES FOR A RELYING PARTY TRUST: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/ee913578(v=ws.11)
[19] SET-ADFSRELYINGPARTYTRUST: https://docs.microsoft.com/en-us/powershell/module/adfs/set-adfsrelyingpartytrust?view=winserver2012r2-ps



By double-clicking the related line, you can edit the issuance transform rules if needed.



Updating the custom policy

The configuration for AD FS as a claims provider in your custom policy is straightforward as a simple four steps process when you'd want to override/extend an already existing relying party information along with its existing user journey (canvas).

This process can be described as follows:

1. Add the missing claim type information in the *TrustFrameworkExtensions.xml* custom policy that comes along with the core templates of the "Starter Pack".
2. Define AD FS as a claims provider in the *TrustFrameworkExtensions.xml* custom policy.
3. Register the AD FS claims provider in an (existing) user journey.

The next sections detail the above.

Adding the missing claim type information

This very first step is optional depending if you'd like to leverage or not one or several additional claim types that do not yet exist in the *TrustFrameworkBase.xml* file that comes along with the core templates of the "Starter Pack".

As discussed in the third document of this series, you can leverage for that purpose the *TrustFrameworkExtensions.xml* policy file. This is illustrated later in this document.

Defining AD FS as a claims provider

You will need at this stage to configure AD FS as a claims provider.

As outlined before, to limit the amount of information to specify in all of your custom policies that may use this claims provider, you can seamlessly leverage the *TrustFrameworkExtensions.xml* policy file of the core templates of the "Starter Pack" to provide a core definition for the AD FS claims provider with an associated technical profile. This technical profile is named **ADFS-WSFED-Outbound** hereafter.

Proceed with the following steps:

1. Navigate to the *SocialAndLocalAccounts* folder in the "Starter Pack".
2. Open the extension policy file, i.e. the *TrustFrameworkExtensions.xml* file, using an XML editor of your choice, for instance Visual Studio (Code).
3. Scroll down to the *ClaimsProviders* section.
4. Add the following *ClaimsProvider* element to the list in the *ClaimsProviders* node. Replace your_adfs_url with the actual FQDN DNS name of your on-premises AD FS server/farm.

```
<ClaimsProvider>
  <Domain>litware369.com</Domain>
  <DisplayName>Litware369 ADFS</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="ADFS-WSFED-Outbound">
      <DisplayName>Litware369 ADFS</DisplayName>
      <Description>Login with your Litware369 account</Description>
      <Protocol Name="WsFed" />
      <Metadata>
        <Item Key="PartnerEntity">https://<your_adfs_domain>/FederationMetadata/2007-
06/FederationMetadata.xml</Item>
      </Metadata>
      <OutputClaims>
       <OutputClaim ClaimTypeReferenceId="userId" PartnerClaimType="employeeNumber" />
       <OutputClaim ClaimTypeReferenceId="givenName" />
       <OutputClaim ClaimTypeReferenceId="surname" />
       <OutputClaim ClaimTypeReferenceId="displayName" PartnerClaimType="CommonName" />
       <OutputClaim ClaimTypeReferenceId="sub" PartnerClaimType="UPN" />
       <OutputClaim ClaimTypeReferenceId="identityProvider" DefaultValue="ADFS" />
```

```
       <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="enterpriseIdp" />
      </OutputClaims>
      <OutputClaimsTransformations>
        <OutputClaimsTransformation ReferenceId="CreateRandomUPNUserName" />
        <OutputClaimsTransformation ClaimTypeReferenceId="CreateUserPrincipalName" />
        <OutputClaimsTransformation ClaimTypeReferenceId="CreateAlternativeSecurityId" />
        <OutputClaimsTransformation ClaimTypeReferenceId="CreateSubjectClaimFromAlternativeSecurityId" />
      </OutputClaimsTransformations>
    </TechnicalProfile>
  </TechnicalProfiles>
<ClaimsProvider>
```

You will not be surprised that the protocol being used here is the **WsFed** (WS-Federation) protocol. It could also have been **SAML2** as ADFS supports it also, see section § *Integrating AD FS as a claims provider via SAML-P 2.0*.

Note For more information, see section § *Specifying a technical profile for a WS-Fed claims provider* in the sixth document of this series, along with the implementation notes for the technical profile if any.

5. Save the policy XML file.

Registering the AD FS claims provider to the Sign-Up or Sign-In (SUSI) user journey

To register the AD FS claims provider to the Sign-Up or Sign-In (SUSI) user journey, proceed with the following steps:

1. Still from the *SocialAndLocalAccounts* folder in the "Starter Pack", open the *TrustFrameworkExtensions.xml* file using an XML editor of your choice.
2. Find the *UserJourneys* element. If the element doesn't exist, add one (or uncomment it).
3. Copy the entire content of *UserJourneys* node from the *TrustFrameworkBase.xml* file as a child of the *UserJourneys* element in the *TrustFrameworkExtensions.xml* file. If you already copied this content, you can skip this step.
4. Find the *UserJourney* element with attribute value Id="SignUpOrSignIn".
5. Locate the *OrchestrationStep* element with attribute Order="1" value under this element.
6. Add the following XML snippet under the *ClaimsProviderSelections* node:

```
<ClaimsProviderSelection TargetClaimsExchangeId="AdfsExchange" />
```

7. Locate the *OrchestrationStep* element with attribute Order="2" in the *UserJourney* node and add the following XML snippet under the *ClaimExchanges* node:

```
<ClaimsExchange Id="AdfsExchange" TechnicalProfileReferenceId="ADFS-WSFED-Outbound" />
```

8. Save the XML file.

Registering the AD FS claims provider to the Profile Edit user journey (Optional)

To register the AD FS claims provider to the Profile Edit user journey, proceed with the following steps:

1. Still from the *SocialAndLocalAccounts* folder in the "Starter Pack", open the *TrustFrameworkExtensions.xml* file using an XML editor of your choice.
2. Find the *UserJourneys* element, and then find the *UserJourney* element with attribute value Id="ProfileEdit".
3. Locate the *OrchestrationStep* element with attribute Order="1" value under this element.
4. Add the following XML snippet under the *ClaimsProviderSelections* node:

```
<ClaimsProviderSelection TargetClaimsExchangeId="AdfsExchange" />
```

5. Locate the *OrchestrationStep* element with attribute Order="2" in the *UserJourney* node and add the following XML snippet under the *ClaimExchanges* node:

```
<ClaimsExchange Id="AdfsExchange" TechnicalProfileReferenceId="ADFS-WSFED-Outbound" />
```

6. Save the XML file.

Uploading the custom policy to your B2C tenant

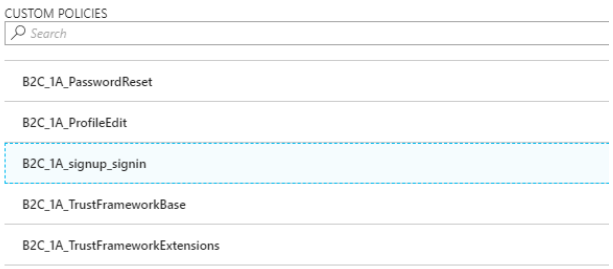
Proceed with the following steps:

1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
2. Select **Upload Policy**.
3. Check **Overwrite the policy if it exists**.
4. In **Upload policy**, select the *TrustFrameworkExtensions.xml* policy file
5. Click **Upload** and ensure that it does not fail the validation.

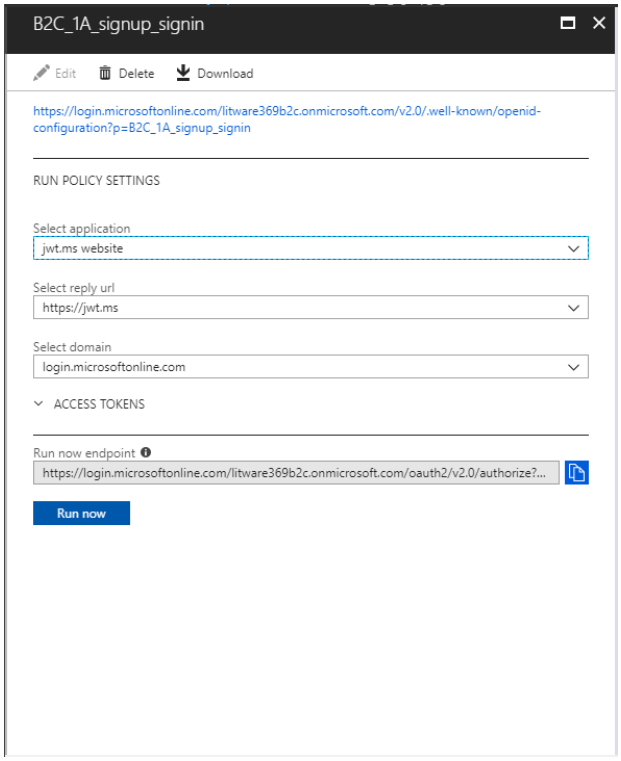
Testing the custom policy by using Run Now

Proceed with the following steps:

1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.



2. Click the **B2C_1A_signup_signin** custom policy to see its detail view. A new blade opens.



3. Click **Run now**.

The sign-in page should include the option to sign in with the identity in the on-premises environment, thanks to the AD FS infrastructure.

Integrating AD FS as a claims provider via SAML-P 2.0

This section shows you, in this use case, how to configure on-premises AD FS as a claims provider (via the SAML-P 2.0 protocol) in your B2C tenant.

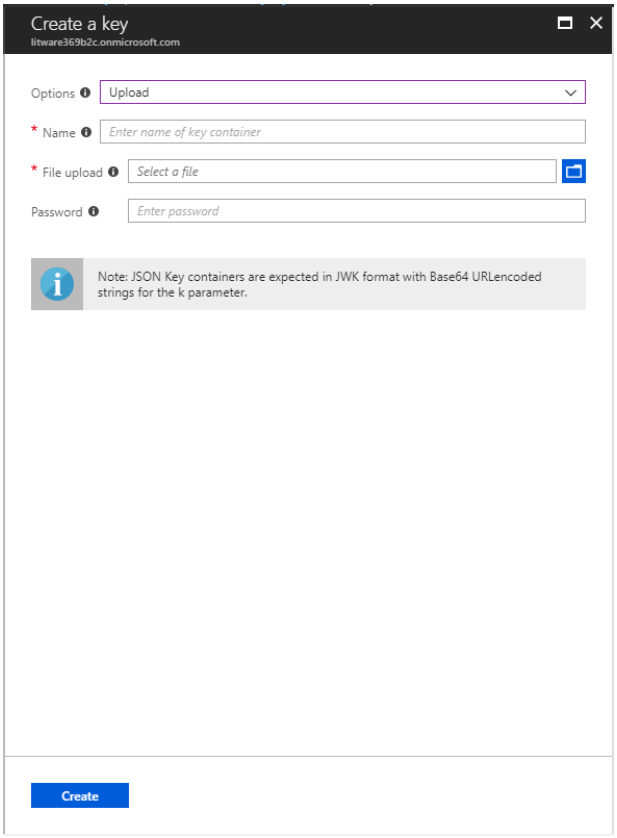
If you're not interested in this use case, you can skip this entire section and jump to the next one.

The configuration steps are similar to ones of the previous section excepted for the relying party trust that slightly defer. You can refer to article AZURE ACTIVE DIRECTORY B2C: ADD ADFS AS A SAML IDENTITY PROVIDER USING CUSTOM POLICIES[20] for specific guidance.

For the configuration, you can leverage the sample policy files provided under the folder *scenarios\aadb2c-ief-setup-adfs2016-app* of the "Starter Pack" for enabling an on-premises AD FS as claims provider in your B2C tenant.

To complete the configuration, you will need a certificate (with its private key) issued from a public certificate authority such as DigiCert. The certificate file (without the private key), i.e. the .cer file, should be added on the AD FS relying party side, while the certificate file (with the private key), i.e. the .pfx file should be added the Identity Experience Framework side.

[20] AZURE ACTIVE DIRECTORY B2C: ADD ADFS AS A SAML IDENTITY PROVIDER USING CUSTOM POLICIES: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-custom-setup-adfs2016-idp



The following XML snippet provide a core definition for the AD FS claims provider with an associated technical profile (via SAML-P 2.0) in the *TrustFrameworkExtensions.xml* policy file of the core templates of the "Starter Pack". This technical profile is named **ADFS-SAML2-Outbound** hereafter.

The key **B2C_1A_ADFSSamlCert** corresponds to the above certificate (with its private key).

```
<ClaimsProvider>
   <Domain>litware369.com</Domain>
   <DisplayName>Litware369 ADFS</DisplayName>
   <TechnicalProfiles>
      <TechnicalProfile Id="ADFS-SAML2-Outbound">
         <DisplayName>Litware369 ADFS</DisplayName>
         <Description>Login with your Litware369 account</Description>
         <Protocol Name="SAML2"/>
         <Metadata>
            <Item Key="RequestsSigned">false</Item>
            <Item Key="WantsEncryptedAssertions">false</Item>
            <Item Key="PartnerEntity">https://<your_ADFS_url>/federationmetadata/2007-
06/federationmetadata.xml</Item>
         </Metadata>
         <CryptographicKeys>
            <Key Id="SamlAssertionSigning" StorageReferenceId="B2C_1A_ADFSSamlCert"/>
            <Key Id="SamlMessageSigning" StorageReferenceId="B2C_1A_ADFSSamlCert"/>
         </CryptographicKeys>
         <OutputClaims>
            <OutputClaim ClaimTypeReferenceId="socialIdpUserId" PartnerClaimType="userPrincipalName" />
            <OutputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="given_name"/>
            <OutputClaim ClaimTypeReferenceId="surname" PartnerClaimType="family_name"/>
            <OutputClaim ClaimTypeReferenceId="email" PartnerClaimType="email"/>
            <OutputClaim ClaimTypeReferenceId="displayName" PartnerClaimType="name"/>
            <OutputClaim ClaimTypeReferenceId="identityProvider" DefaultValue="contoso.com" />
            <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="socialIdpAuthentication"/>
         </OutputClaims>
         <OutputClaimsTransformations>
```

```
            <OutputClaimsTransformation ReferenceId="CreateRandomUPNUserName"/>
            <OutputClaimsTransformation ReferenceId="CreateUserPrincipalName"/>
            <OutputClaimsTransformation ReferenceId="CreateAlternativeSecurityId"/>
            <OutputClaimsTransformation ReferenceId="CreateSubjectClaimFromAlternativeSecurityId"/>
        </OutputClaimsTransformations>
        <UseTechnicalProfileForSessionManagement ReferenceId="SM-Noop"/>
      </TechnicalProfile>
   </TechnicalProfiles>
</ClaimsProvider>
```

Note For more information, see section § *Specifying a technical profile for a SAML 2.0 claims provider* in the sixth document of this series, along with the implementation notes for the technical profile if any.

Integrating Salesforce as a claims provider

This section shows how to integrate the Salesforce IDP as an example of a SAML-P 2.0 claims provider.

Note For more information, see article AZURE ACTIVE DIRECTORY B2C: SIGN IN BY USING SALESFORCE ACCOUNTS VIA SAML[21].

If you're not interested in this use case, you can skip this entire section and jump to the next one.

To do so, you will have to:

1. Setup a My Domain in Salesforce.
2. Setup the identity provider in Salesforce.
3. Create a Salesforce application.
4. Create a signed certificate for your B2C tenant.
5. Add a SAML signing certificate to your B2C tenant.
6. Update custom policies to use Salesforce as claim provider in the intended user journey(s).
7. Upload custom policies.
8. Test the custom policy using **Run Now**.

The next sections detail the above.

If you don't have a Salesforce developer account you can sign-up for a free developer account at https://developer.salesforce.com/signup, and then setup a My Domain Name.

This is the purpose of the next section. We assume you are using Salesforce Lightning Experience[22].

Setting up a My Domain Name in Salesforce

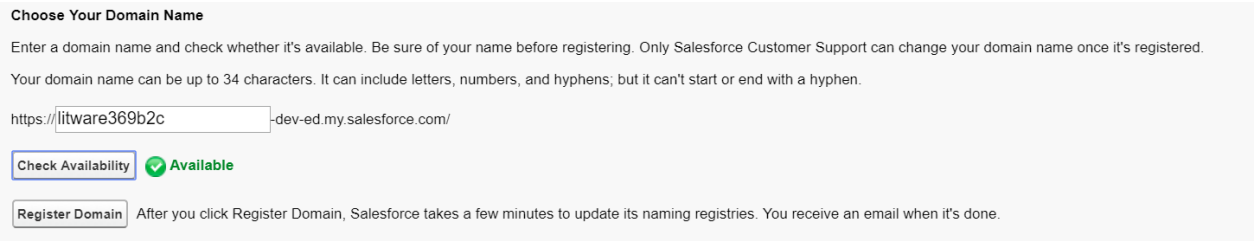
Proceed with the following steps:

Note For more information, see article SET UP A MY DOMAIN NAME[23].

1. Login to your Salesforce developer account.
2. In the left menu, under **Settings**, select **Company Settings**, then click **My Domain**.

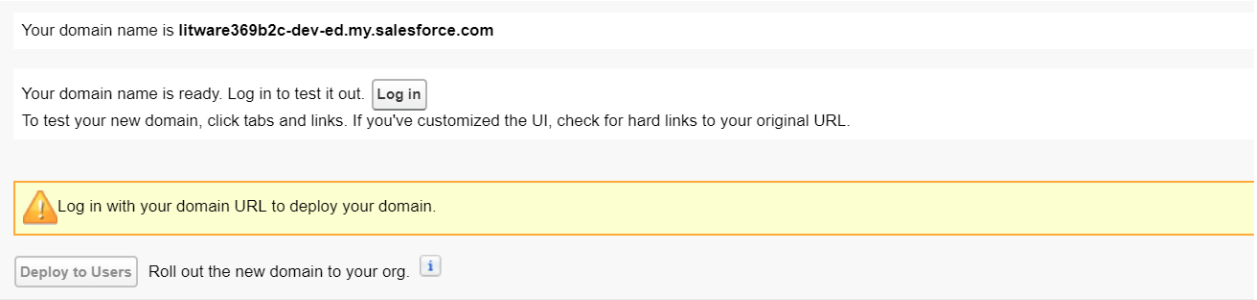
[21] AZURE ACTIVE DIRECTORY B2C: SIGN IN BY USING SALESFORCE ACCOUNTS VIA SAML: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-setup-sf-app-custom
[22] LIGHTNING EXPERIENCE FAQ: https://developer.salesforce.com/page/Lightning_Experience_FAQ
[23] SET UP A MY DOMAIN NAME: https://help.salesforce.com/articleView?id=domain_name_setup.htm&type=0

3. Under **Choose Your Domain Name**, type your domain name, for example “litware369b2c” in our illustration.
4. Click **Check Availability**. If the domain name is unique, it will be available.

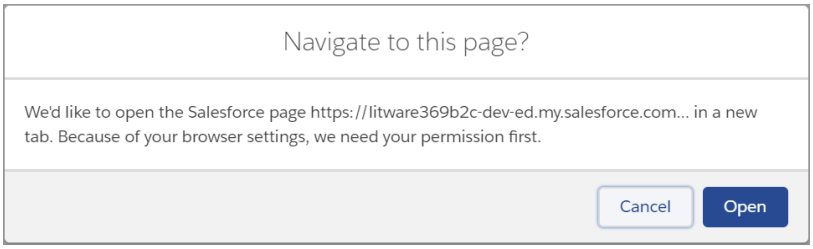


5. Click **Register Domain**. Please **wait 5-10 minutes** before proceeding to the next step. The custom domain name you just registered requires some time to take effect.

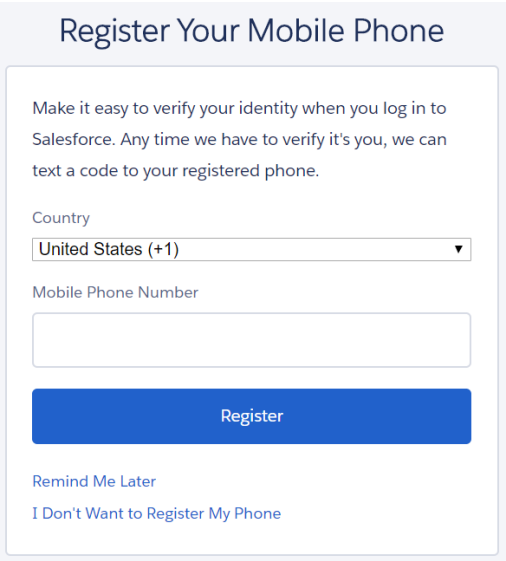
   Refresh the page. If you don’t see Authentication Configuration section, wait another minute and try again.



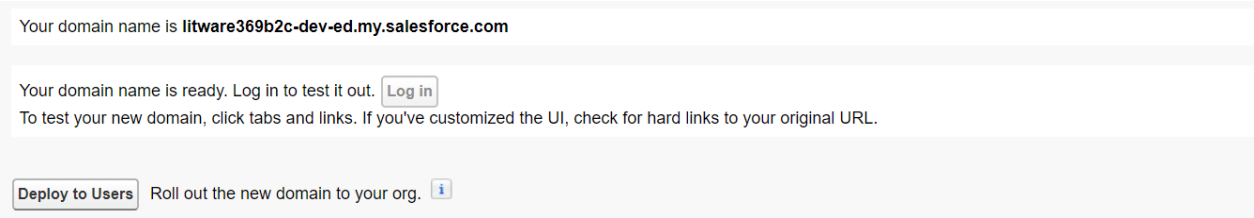
6. Click **Log in**.



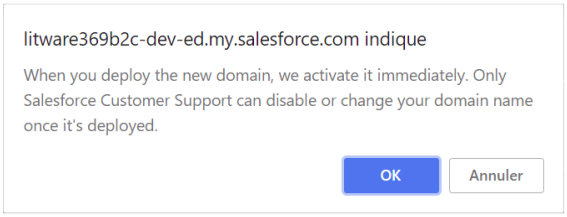
7. Click **Open**. A new tab opens.



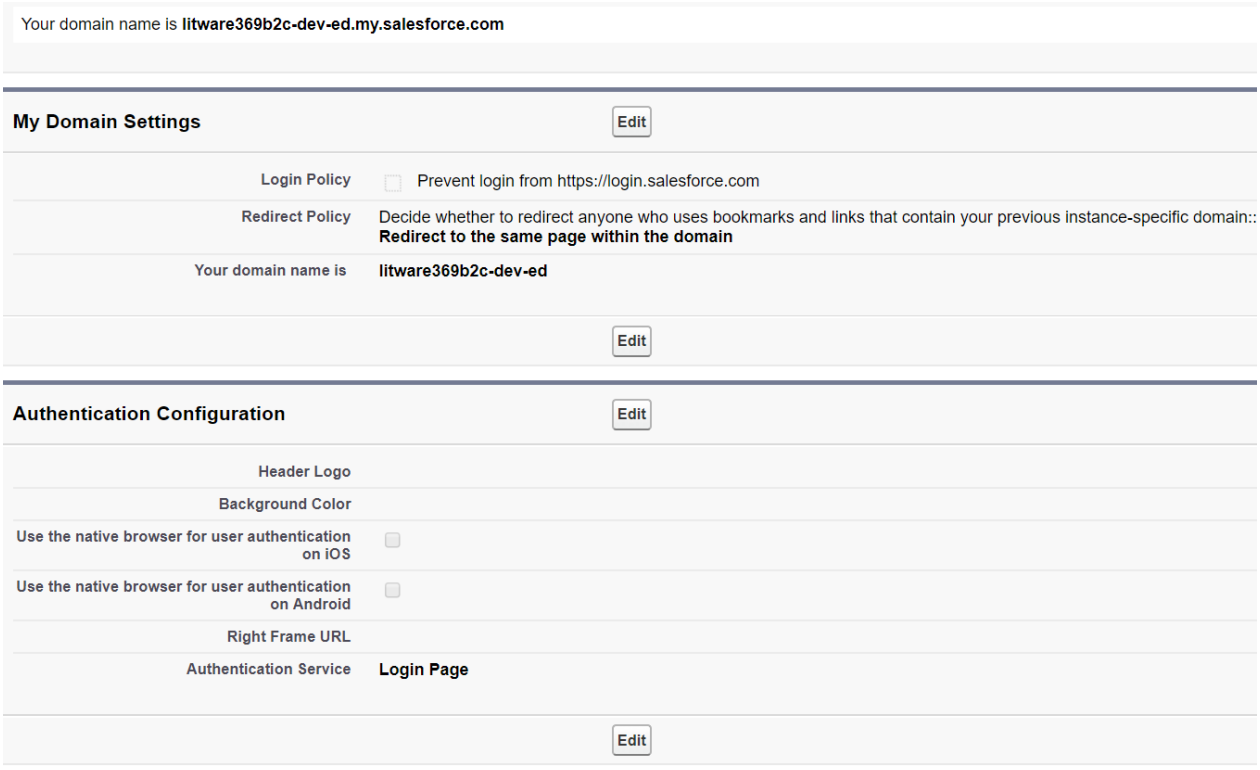
8. Click **I Don’t Want to Register My Phone**. The tab closes.



9. Click **Deploy to Users**.



10. Click **OK** on the dialog box to confirm domain deployment.



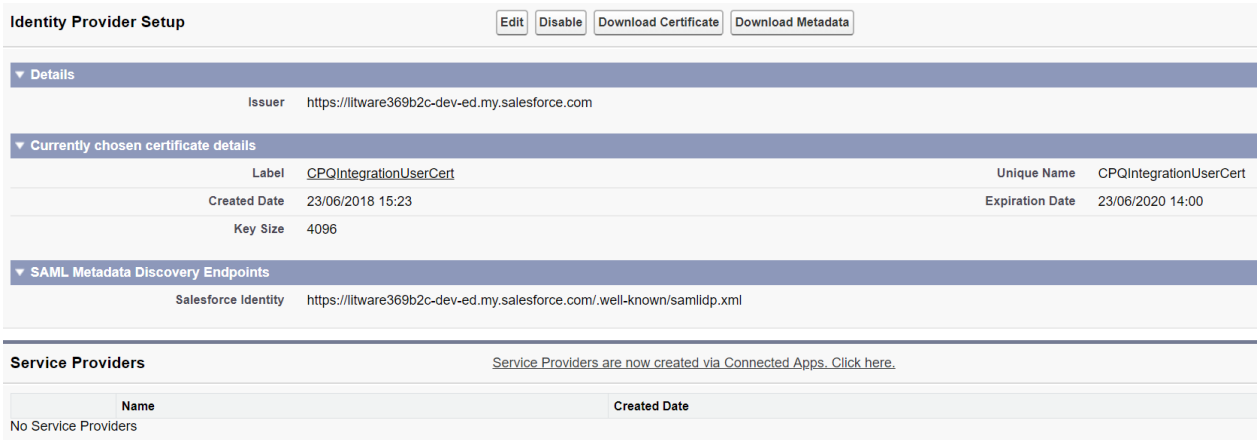
11. You can now sign-in to https://litware369b2c-dev-ed.my.salesforce.com.

Setting up the identity provider in Salesforce

Proceed with the following steps:

1. In the left menu, under **Settings**, expand the **Identity** menu and then select **Identity Provider**.
2. Select **Enable Identity Provider** to enable Salesforce as an identity provider.

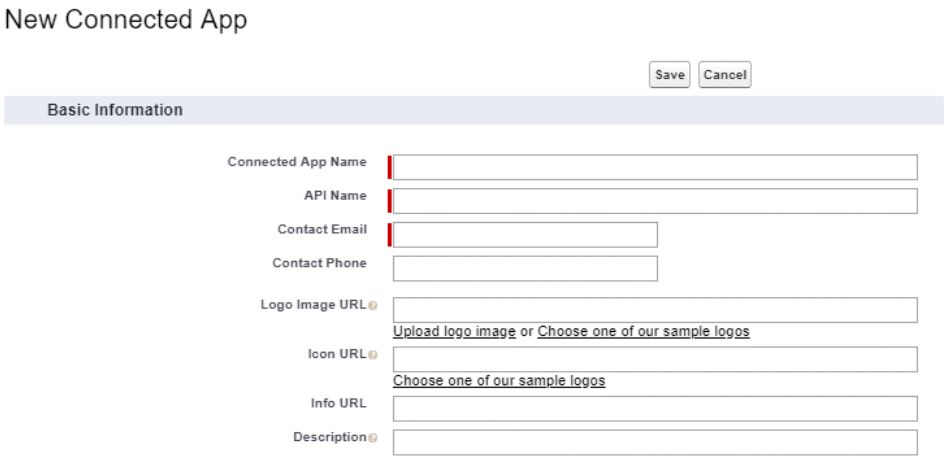
3. Select the default certificate from the list and then select **Save**.



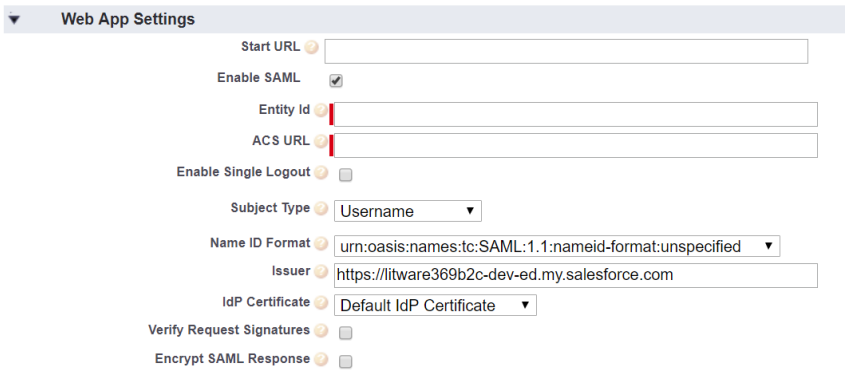
Creating a Salesforce application in Salesforce

Proceed with the following steps:

1. On the **Identity Provider** page, select the Service Providers link **Service Providers are now created via Connected Apps. Click here**. A **New Connected App** page opens.



2. Provide a connected app name, API name, and contact email.
3. Scroll down to the **Web App Settings** section and select **Enable SAML**.



4. In **Entity Id**, enter:

https://login.microsoftonline.com/te/<*your_b2c_tenant*>.onmicrosoft.com/<*your_base_policy*>B2C_1A_TrustFrameworkBase

where *your_b2c_tenant* is the name of your B2C tenant, and *your_base_policy* the base policy the custom policy being executed inherits from. For example, in our configuration:

https://login.microsoftonline.com/te/litware369b2c.onmicrosoft.com/B2C_1A_TrustFrameworkBase

5. In **ACS URL**, enter:

https://login.microsoftonline.com/te/<*your_b2c_tenant*>.onmicrosoft.com/<*your_custom_policy*>/SAMLP/SSO/ASSERTIONCONSUMER

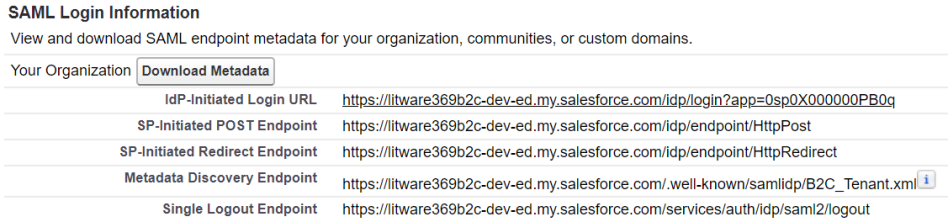
where *your_b2c_tenant* is the name of your B2C tenant, and *your_custom_policy* the custom policy being executed. For example, in our configuration:

https://login.microsoftonline.com/te/litware369b2c.onmicrosoft.com/B2C_1A_Signup_Signin/SAMLP/SSO/ASSERTIONCONSUMER

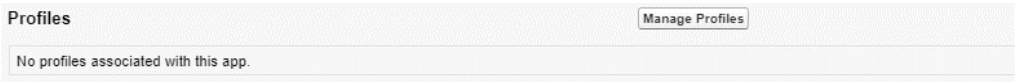
6. Leave the remaining fields at their default values, scroll down, and select **Save.**
7. On the overview page for the new app, select **Manage**.

8. Scroll down to the **SAML Login Information** section. Record the value for **Metadata Discovery Endpoint**. For example, in our configuration:

https://litware369b2c-dev-ed.my.salesforce.com/.well-known/samlidp/B2C_Tenant.xml



9. Scroll down to the **Profiles** section, and then select **Manage Profile**.



10. Select the profile or group you want to federate with Azure AD B2C. Select **System Administrator** so that you can federate with your account.
11. Click **Save**.

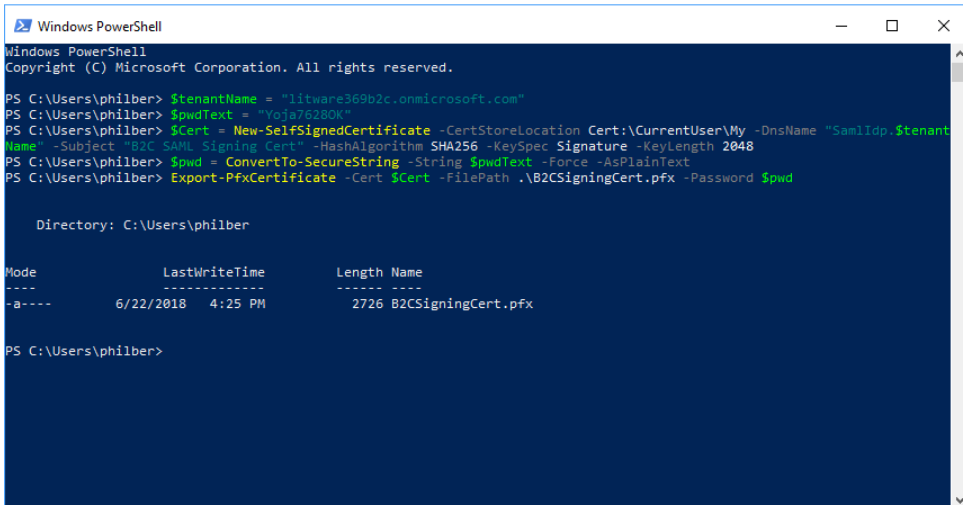
Generating a signing certificate for your B2C tenant

Requests sent to Salesforce need to be signed by your B2C tenant.

To generate a signing certificate, proceed with the following steps:

1. Open a PowerShell command prompt.
2. Run the following commands. Replace *<your_b2c_tenant>* with the name of your B2C tenant, for example **litware369b2c** in our configuration, and provide a password for the certificate.

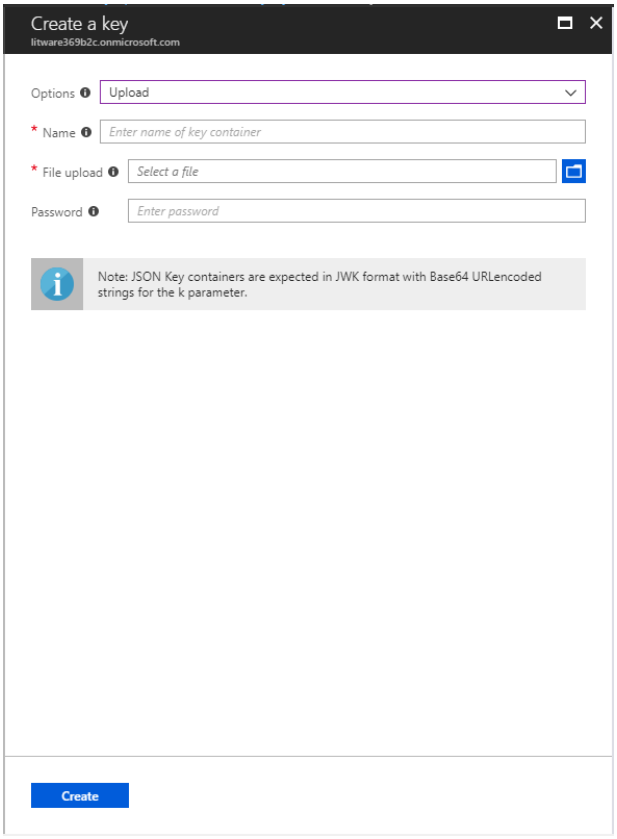
```
$tenantName = "<you_b2c_tenant>.onmicrosoft.com"
$pwdText = "<your_password here>"

$Cert = New-SelfSignedCertificate -CertStoreLocation Cert:\CurrentUser\My -DnsName
"SamlIdp.$tenantName" -Subject "B2C SAML Signing Cert" -HashAlgorithm SHA256 -KeySpec Signature -KeyLength 2048

$pwd = ConvertTo-SecureString -String $pwdText -Force -AsPlainText

Export-PfxCertificate -Cert $Cert -FilePath .\B2CSigningCert.pfx -Password $pwd
```



Adding a SAML signing certificate to your B2C tenant

Proceed with the following steps:

1. In the Azure portal, go to the **Azure AD B2C** blade, and then select **Identity Experience Framework - PREVIEW**.
2. Select **Policy Keys**.
3. Select **+Add**.
4. In the **Options** list, select **Upload**.



5. Set the **Name** to "SAMLSigningCert".
6. Upload the *B2CSigningCert.pfx* file you created in the in the previous procedure and provide the password that you specified when you created the certificate.
7. Select **Create**.
8. When the key has been created, record the key full name, including the **B2C_1A** prefix (it should be something like **B2C_1A_SAMLSigningCert** if you specified the **Subject** of the certificate exactly as shown previously).

Updating custom policies

Define Salesforce as claim provider

To register Salesforce as claim provider in your custom policy files as follows, proceed with the following steps:

1. Navigate to the *SocialAndLocalAccounts* folder in the "Starter Pack".
2. Open the extension policy file, i.e. the *TrustFrameworkExtensions.xml* file, using an XML editor of your choice, for instance Visual Studio (Code).
3. Scroll down to the *ClaimsProviders* section.

4. Add the following *ClaimsProvider* element to the list in the *ClaimsProviders* node.

```
<ClaimsProvider>

   <!--Domain value must be unique -->
   <Domain>your_Salesforce_domain</Domain>

   <DisplayName>Salesforce</DisplayName>
   <TechnicalProfiles>
      <!-- Technical profile Id will be used to refere this profile in other parts of Policy-->
      <TechnicalProfile Id="Salesforce-SAML2-outbound">
         <!-- Sign in button text-->
         <DisplayName>Salesforce</DisplayName>
         <Description>Login with your Salesforce account</Description>
         <!-- SalesForce use SAML 2.0 so make sure use SAML2 in the value -->
         <Protocol Name="SAML2"/>
         <Metadata>
            <Item Key="RequestsSigned">false</Item>
            <Item Key="WantsEncryptedAssertions">false</Item>
            <Item Key="WantsSignedAssertions">false</Item>

            <!-- Copy the endpoint url here from SalesForce app you have created in SalesForce from metadata section of App
                 overview-->
            <Item Key="PartnerEntity">your_SalesForce_app_endpoint_uri</Item>

         </Metadata>
         <CryptographicKeys>
            <!--This is the key you have created in Azure AD B2C -> Identify Framework Experience->Policy Keys section --
>
            <Key Id="SamlAssertionSigning" StorageReferenceId="B2C_1A_SAMLSigningCert"/>
            <Key Id="SamlMessageSigning" StorageReferenceId="B2C_1A_SAMLSigningCert"/>
         </CryptographicKeys>
         <OutputClaims>
            <OutputClaim ClaimTypeReferenceId="socialIdpUserId" PartnerClaimType="userId"/>
            <OutputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="given_name"/>
            <OutputClaim ClaimTypeReferenceId="surname" PartnerClaimType="family_name"/>
            <OutputClaim ClaimTypeReferenceId="email" PartnerClaimType="email"/>
            <OutputClaim ClaimTypeReferenceId="displayName" PartnerClaimType="username"/>
            <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="externalIdp"/>
            <OutputClaim ClaimTypeReferenceId="identityProvider" DefaultValue="SAMLIdp" />
         </OutputClaims>
         <OutputClaimsTransformations>
            <OutputClaimsTransformation ReferenceId="CreateRandomUPNUserName"/>
            <OutputClaimsTransformation ReferenceId="CreateUserPrincipalName"/>
            <OutputClaimsTransformation ReferenceId="CreateAlternativeSecurityId"/>
            <OutputClaimsTransformation ReferenceId="CreateSubjectClaimFromAlternativeSecurityId"/>
         </OutputClaimsTransformations>
         <UseTechnicalProfileForSessionManagement ReferenceId="SM-Noop"/>
      </TechnicalProfile>
   </TechnicalProfiles>
</ClaimsProvider>
```

Note For more information, see section § *Specifying a technical profile for a SAML 2.0 claims provider* in the sixth document of this series, along with the implementation notes for the technical profile if any.

5. In the *Domain* element, replace your_Salesforce_domain with the name of the domain you created for your Salesforce developer account. For example, **litware369** in our illustration.
6. In the *PartnerEntity* element, replace *your_SalesForce_app_endpoint_uri* with the Salesforce Metadata Discovery Endpoint that you recorded earlier. For example, in our configuration:

https://litware369-dev-ed.my.salesforce.com/.well-known/samlidp/B2C_Tenant.xml

7. Save the XML file.

Registering the Salesforce claims provider to the Sign-Up or Sign-In (SUSI) user journey

To register the Salesforce claims provider to the Sign-Up or Sign-In (SUSI) user journey, proceed with the following steps:

1. Still from the *SocialAndLocalAccounts* folder in the "Starter Pack", open the *TrustFrameworkExtensions.xml* file using an XML editor of your choice.
2. Find the *UserJourneys* element. If the element doesn't exist, add one (or uncomment it).
3. Copy the entire content of *UserJourneys* node from the *TrustFrameworkBase.xml* file as a child of the *UserJourneys* element in the *TrustFrameworkExtensions.xml* file. If you already copied this content, you can skip this step.
4. Find the *UserJourney* element with attribute value Id="SignUpOrSignIn".
5. Locate the *OrchestrationStep* element with attribute Order="1" value under this element.
6. Add the following XML snippet under the *ClaimsProviderSelections* node:

```
<ClaimsProviderSelection TargetClaimsExchangeId="SalesforceExchange" />
```

7. Locate the *OrchestrationStep* element with attribute Order="2" in the *UserJourney* node and add the following XML snippet under the *ClaimExchanges* node:

```
<ClaimsExchange Id="SalesforceExchange" TechnicalProfileReferenceId="Salesforce-SAML2-Outbound" />
```

8. Save the XML file.

Uploading the custom policy to your B2C tenant

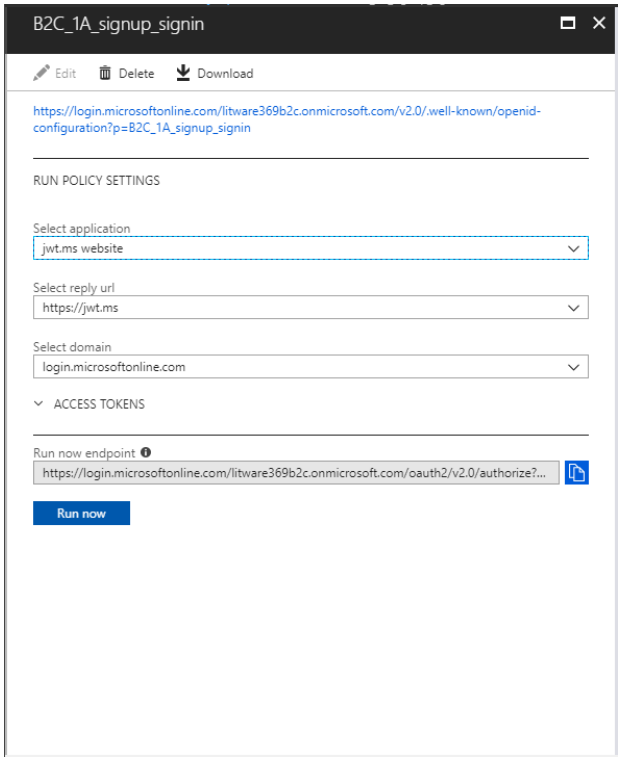
Proceed with the following steps:

1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
2. Select **Upload Policy**.
3. Check **Overwrite the policy if it exists**.
4. In **Upload policy**, select the *TrustFrameworkExtensions.xml* policy file
5. Click **Upload** and ensure that it does not fail the validation.

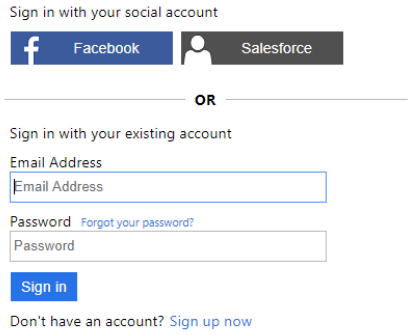
Testing the custom policy by using Run Now

Proceed with the following steps:

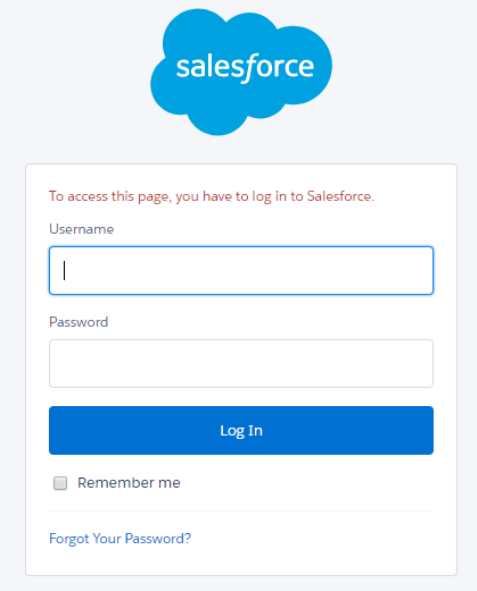
1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
2. Click the **B2C_1A_signup_signin** custom policy to see its detail view. A new blade opens.



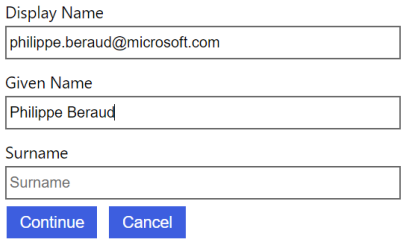
3. Click **Run now**. The sign-in page should include the option to sign in with Salesforce.



4. You should be able to sign in using your Salesforce credentials.



Enter your Salesforce credential and clock **Log In**.



5. Complete the sign-up process and click **Continue**. The received claims are rendered by the https://jwt.ms website. You can see the IdP claim set to SAMLIdP.

Configuring password complexity for local accounts

Azure AD B2C uses by default strong passwords for local accounts. However, the custom policies support changing the complexity requirements for passwords supplied by an end user when creating an account. This is done at the custom policy level. As such, in other words, it is possible to have one custom policy

require a four-digit pin during sign-up while another custom policy requires an eight-character string during sign-up. For example, you may use a policy with different password complexity for adults than for children.

If you're not interested in this use case, you can skip this entire section and jump to the next part.

This section shows how to configure the password complexity.

> **Note** For more information, see article CONFIGURE PASSWORD COMPLEXITY IN B2C WITH CUSTOM POLICIES[24].

To do so, you will have to:

1. Update the custom policy to configure the password complexity.
2. Upload the custom policy.
3. Test the custom policy using **Run Now**.

The next sections detail the above.

Updating the custom policy

To configure password complexity in custom policy, proceed with the following steps:

1. Navigate to the *SocialAndLocalAccounts* folder in the "Starter Pack".
2. Open the Sign-Up or Sign-In (SUSI) relying party (RP) policy file, i.e. the *SignUpOrSignIn.xml* file, using an XML editor of your choice, for instance Visual Studio (Code).
3. Between the </BasePolicy> and the <RelyingParty> elements, add the following XML snippet.

```
<BuildingBlocks>
   <ClaimsSchema>
      <ClaimType Id="newPassword">
         <InputValidationReference Id="PasswordValidation" />
      </ClaimType>
      <ClaimType Id="reenterPassword">
          <InputValidationReference Id="PasswordValidation" />
      </ClaimType>
   </ClaimsSchema>
   <Predicates>
      <Predicate Id="Lowercase" Method="MatchesRegex" HelpText="a lowercase">
         <Parameters>
            <Parameter Id="RegularExpression">[a-z]</Parameter>
         </Parameters>
      </Predicate>
      <Predicate Id="Uppercase" Method="MatchesRegex" HelpText="an uppercase">
         <Parameters>
            <Parameter Id="RegularExpression">[A-Z]</Parameter>
         </Parameters>
      </Predicate>
      <Predicate Id="Number" Method="MatchesRegex" HelpText="a number">
         <Parameters>
            <Parameter Id="RegularExpression">[0-9]</Parameter>
         </Parameters>
      </Predicate>
      <Predicate Id="Symbol" Method="MatchesRegex" HelpText="a symbol">
         <Parameters>
            <Parameter Id="RegularExpression">[!@#$%^*()]</Parameter>
         </Parameters>
      </Predicate>
      <Predicate Id="Length" Method="IsLengthRange" HelpText="The password must be between 8 and 16 characters.">
         <Parameters>
            <Parameter Id="Minimum">8</Parameter>
            <Parameter Id="Maximum">16</Parameter>
         </Parameters>
```

[24] CONFIGURE PASSWORD COMPLEXITY IN B2C WITH CUSTOM POLICIES: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-reference-password-complexity-custom

```
      </Predicate>
   </Predicates>
   <InputValidations>
      <InputValidation Id="PasswordValidation">
         <PredicateReferences Id="LengthGroup" MatchAtLeast="1">
            <PredicateReference Id="Length" />
         </PredicateReferences>
         <PredicateReferences Id="3of4" MatchAtLeast="3"
                                        HelpText="You must have at least 3 of the following character classes:">
            <PredicateReference Id="Lowercase" />
            <PredicateReference Id="Uppercase" />
            <PredicateReference Id="Number" />
            <PredicateReference Id="Symbol" />
         </PredicateReferences>
      </InputValidation>
   </InputValidations>
</BuildingBlocks>
```

The above XML snippet defines an input validation rule for strong passwords. This rule validates that a password is 8 to 16 characters and contains 3 of 4 of numbers, uppercase, lowercase, or symbols.

For that purpose, it adds, inside the *BuildingBlocks* node, *Predicates*, *InputValidations*, and *ClaimsSchema* elements to the policy. The purpose of these elements is as follows:

- The *Predicates* element has a series of *Predicate* elements, each of them defining a basic string validation check that returns true or false.
- The *InputValidations* element has one or more *InputValidation* elements. Each *InputValidation* is constructed by using a series of *Predicate* elements. This element allows you to perform boolean aggregations (similar to **and** and **or**).
- The *ClaimsSchema* defines which claim is being validated. It then defines which *InputValidation* rule is used to validate that claim.

As far as the latter is concerned, the above claim types **newPassword** and **reenterPassword** are considered special, so do not change the names. The UI validates the user correctly reentered their password during account creation based on these *ClaimType* elements. The same *ClaimType* elements are found in the **B2C_1A_TrustFrameworkBase** policy from which inherits this policy.

These elements' definitions are here overridden to define an *InputValidationReference* element. The *ID* attribute of this new element is pointing to the *InputValidation* element that we define at the end of the XML snippet.

An *InputValidation* element is an aggregation of *PredicateReferences*. Each *PredicateReferences* must be true in order for the *InputValidation* to succeed. However, inside the *PredicateReferences* element use an attribute called *MatchAtLeast* to specify how many *PredicateReference* checks must return true. Optionally, define a *HelpText* attribute to override the error message defined in the *Predicate* elements that it references.

Eventually, the *InputValidation* element is referenced in *ClaimType* elements, here **newPassword** and **reenterPassword**. Therefore, this above rule is being enforced in this custom policy.

4. Save the XML file.

Uploading the custom policy to your B2C tenant

Proceed with the following steps:

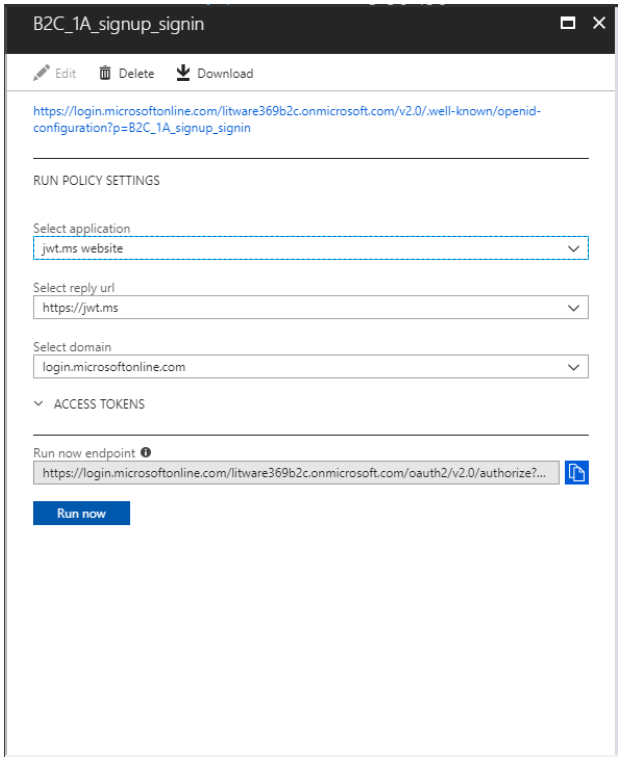
1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
2. Select **Upload Policy**.

3. Check **Overwrite the policy if it exists**.
4. In **Upload policy**, select the *SignUpOrSignIn.xml* policy file
5. Click **Upload** and ensure that it does not fail the validation.

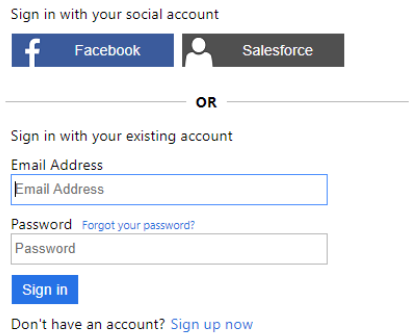
Testing the custom policy by using Run Now

Proceed with the following steps:

1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
2. Click the **B2C_1A_signup_signin** custom policy to see its detail view. A new blade opens.

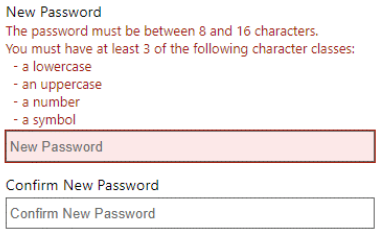


3. Click **Run now**.



4. Click **Sign up now**.
5. Click **Send verification code**.
6. Enter the received code in the Verification code and click **Verify Code**.

7. Select **New Password**.



The expected input validation is in place. Type a password. As the specified password respects the predicates, these are erased from the above message in red.

Bringing your own identity (BYOI) for social users

This section illustrates how to use in a non-exhaustive manner popular external social identity providers (IdPs) to integrate with Azure AD B2C in order to "*Bring Your Own Identity*" (BYOI) for social user to help you:

- Collecting additional scopes from Facebook.
- Integrating Google+ as a claims provider.
- Integrating Amazon as a claims provider.
- Integrating LinkedIn as a claims provider.
- Integrating Microsoft Account (MSA) as a claims provider.
- Integrating Twitter as a claims provider.
- Pre-filling a request with a domain hint.

The following subsections depicts the related configuration for your B2C tenant and your custom policies based on the "Starter Pack". They assume that you followed the instructions provided in the second document of this series to setup and configure the "Starter Pack".

Collecting additional scopes from Facebook

The second document of this series provides instructions on how to declare Facebook as a social identity provider in your B2C tenant and how to use it with the custom policies that comes with the "Starter Pack".

At this stage, Facebook should be already configured in your B2C tenant as a social identity provider.

This section shows you how to collect additional scopes from Facebook. **If you're not interested in this use case, you can skip this entire section and jump to the next one.**

To do so, you will have to:

1. Review the available scopes from Facebook.
2. Update the technical profile of the Facebook claims provider and the user journey(s) in custom policies to include the additional scopes.
3. Upload custom policies.
4. Test the custom policy using **Run Now**.

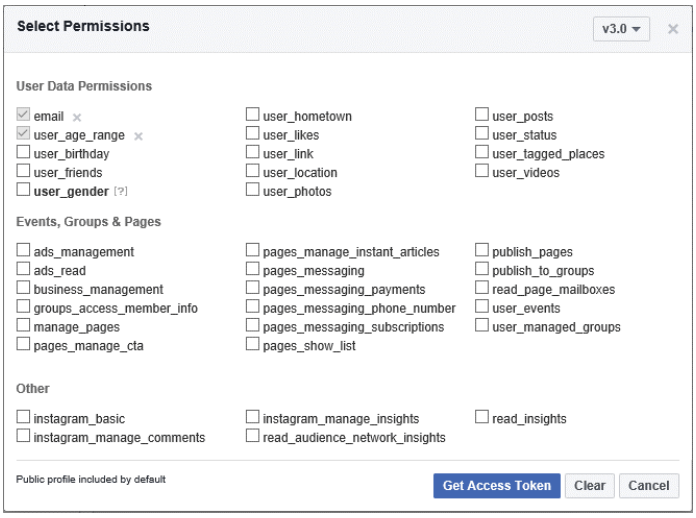
The next sections detail the above.

Reviewing the available scope from Facebook

You can leverage the Graph API Explorer tool from Facebook to see and query the available scope.

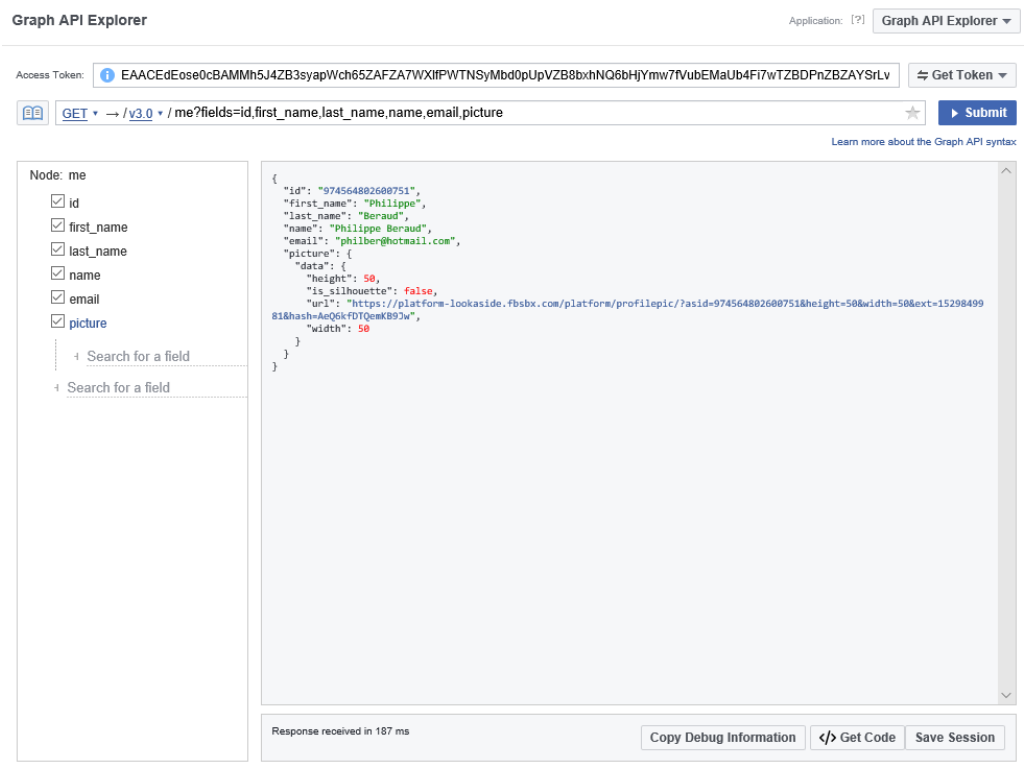
Proceed with the following steps:

1. Open a browsing session, navigate to https://developers.facebook.com/tools/explorer/ and then click **Log In** to sign-in with your Facebook account.
2. Click **Get Token** and then **Get User Access Token** to obtain a suitable access token. A Select Permissions dialog pops up.



   a. Select the required **User Data Permissions** based on the information you'd like to retrieve.
   b. Click **Get Access Token**.
3. Enter your query, for example "*me?fields=id,first_name,last_name,name,email,picture*" in our illustration and click **Submit**.

   For the sake of the illustration, we are interested in retrieving the URL of the user picture in addition to the other fields already configured in the Facebook claims provider.



Updating custom policies

As per the second document in this series, Facebook is already defined in the custom policy as a claims provider with an associated technical profile. This technical profile is named **Facebook-OAUTH** hereafter.

To obtain additional scopes, proceed with the following steps:

1. Navigate to the *SocialAndLocalAccounts* folder in the "Starter Pack".
2. Open the extension policy file, i.e. the *TrustFrameworkExtensions.xml* file, using an XML editor of your choice, for instance Visual Studio (Code).
3. Scroll down to the *BuildingBlocks* section.
4. Add the following XML snippet within the node.

```
<BuildingBlocks>
  <ClaimsSchema>
    <ClaimType Id="picture">
      <DisplayName>Picture</DisplayName>
      <DataType>string</DataType>
    </ClaimType>
  </ClaimsSchema>
</BuildingBlocks>
```

5. Scroll down to the *ClaimsProviders* section.
6. Locate the *ClaimsProvider* element in the *ClaimsProviders* node whose display name is Facebook.

7. Add both the **picture** field to the "ClaimsEndpoint" metadata item and the **picture** output claim to the **Facebook-OAUTH** technical profile as illustrated

```
<!-- Facebook claims provider -->
<ClaimsProvider>
   <DisplayName>Facebook</DisplayName>
   <TechnicalProfiles>
      <TechnicalProfile Id="Facebook-OAUTH">
         <Metadata>
            <Item Key="client_id">462526187284464</Item>
            <Item Key="scope">email public_profile</Item>

            <Item
Key="ClaimsEndpoint">https://graph.facebook.com/me?fields=id,first_name,last_name,name,email,philber</Item>

         </Metadata>

         <OutputClaims>
            <OutputClaim ClaimTypeReferenceId="picture" PartnerClaimType="picture" />
         </OutputClaims>

      </TechnicalProfile>
   </TechnicalProfiles>
</ClaimsProvider>
```

Note For more information, see section § *Specifying a technical profile for an OAuth 2.0 claims provider* in the sixth document of this series, along with the implementation notes for the technical profile if any.

8. Save the XML file.
9. Now open the Sign-Up or Sign-In (SUSI) relying party (RP) policy file, i.e. the *SignUpOrSignIn.xml* file.
10. Scroll down to the *TechnicalProfile* element of the *RelyingParty* node.
11. Add the following XML snippet within the *OutputClaims* node.

```
<OutputClaim ClaimTypeReferenceId="picture" />
```

The *RelyingParty* node should be as follows:

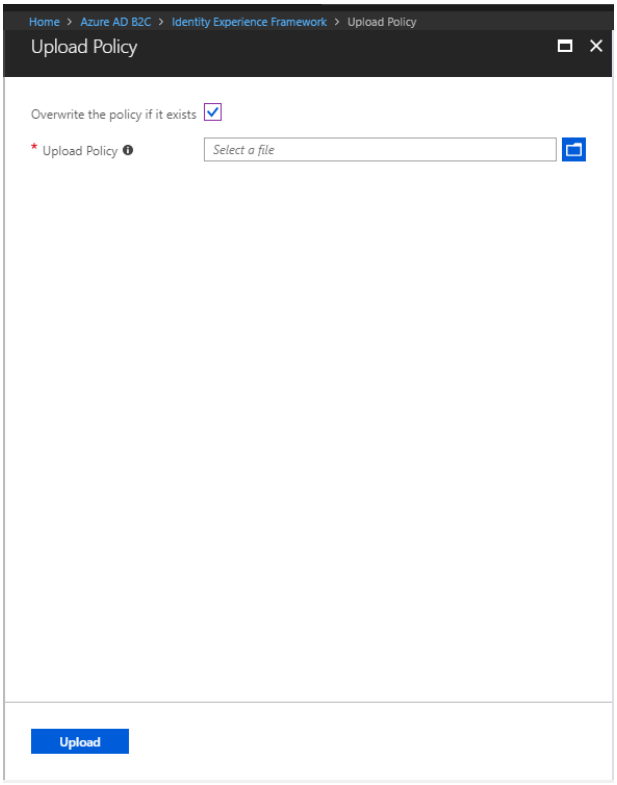
```
<RelyingParty>
   <DefaultUserJourney ReferenceId="SignUpOrSignIn" />
   <TechnicalProfile Id="PolicyProfile">
      <DisplayName>PolicyProfile</DisplayName>
      <Protocol Name="OpenIdConnect" />
      <OutputClaims>
         <OutputClaim ClaimTypeReferenceId="displayName" />
         <OutputClaim ClaimTypeReferenceId="givenName" />
         <OutputClaim ClaimTypeReferenceId="surname" />
         <OutputClaim ClaimTypeReferenceId="email" />

         <OutputClaim ClaimTypeReferenceId="picture" />

         <OutputClaim ClaimTypeReferenceId="objectId" PartnerClaimType="sub" />
         <OutputClaim ClaimTypeReferenceId="identityProvider" />
      </OutputClaims>
      <SubjectNamingInfo ClaimType="sub" />
   </TechnicalProfile>
</RelyingParty>
```

12. Save the XML file.

Uploading custom policies to your B2C tenant

Proceed with the following steps:

1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
2. Select **Upload Policy**.
3. Check **Overwrite the policy if it exists**.

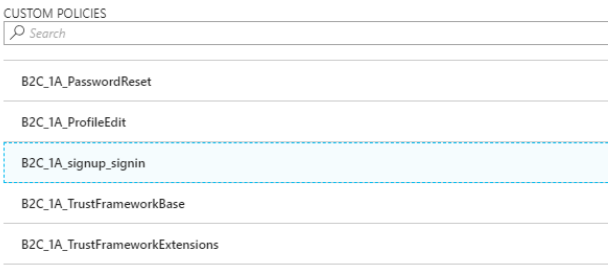


4. In **Upload policy**, select the *TrustFrameworkExtensions.xml* policy file
5. Click **Upload** and ensure that it does not fail the validation.
6. Repeat the above steps with the *SignUpOrSignIn.xml* policy file.

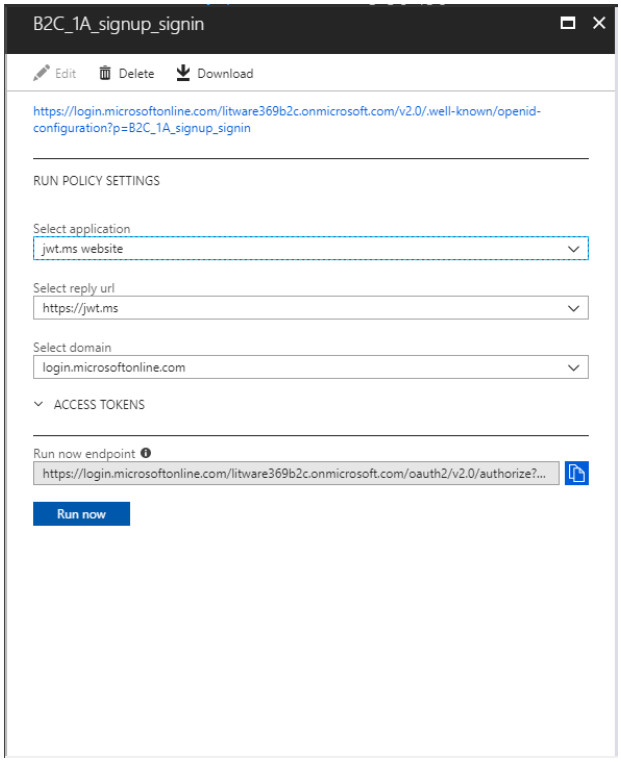
Testing the custom policy by using Run Now

Proceed with the following steps:

1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.



2. Click the **B2C_1A_signup_signin** custom policy to see its detail view. A new blade opens.



3. Click **Run now**.

The sign-in page should include the option to sign in with Facebook. You should be able to sign in using your Facebook account and retrieve in the issued token the picture information.

Decoded Token | Claims


```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid": "1dZZY23T1zabgVVbFSXB__xMlzUCG3XsYV5yIjeJmmU"
}.{
  "exp": 1529601318,
  "nbf": 1529597718,
  "ver": "1.0",
  "iss": "https://login.microsoftonline.com/cbf80679-8d13-4ac7-a5de-a4c1d69bd5d2/v2.0/",
  "sub": "bfdda392-6994-4ce0-93bd-fa4d062f408f",
  "aud": "95b7953b-a499-48b9-bc1b-48bb095d6939",
  "acr": "b2c_1a_signup_signin",
  "nonce": "defaultNonce",
  "iat": 1529597718,
  "auth_time": 1529597718,
  "picture": "{ \"data\": { \"height\": 50, \"is_silhouette\": false, \"url\": \"https://platform-
lookaside.fbsbx.com/platform/profilepic/?asid=964916140232284&height=50&width=50&ext=1529856914&hash=AeRQ68usA_g5xIlD\", \"width\": 50
} }",
  "given_name": "Philippe",
  "family_name": "Beraud",
  "name": "Philippe Beraud",
  "email": "philber@hotmail.com",
  "idp": "facebook.com"
}.[Signature]
```


The next sections cover how to leverage the most common social identity providers for your identity use cases.

Integrating Google+ as a claims provider

This section shows you, in this use case, how to configure Google+ as a claims provider in your B2C tenant.

> **Note** For more information, see article AZURE ACTIVE DIRECTORY B2C: ADD GOOGLE+ AS AN OAUTH2 IDENTITY PROVIDER USING CUSTOM POLICIES[25].

If you're not interested in this use case, you can skip this entire section and jump to the next one.

To do so, you will have to:

1. Create a Google+ application and supply it with the right parameters.
2. Add the Google+ application key in your B2C tenant.
3. Update the custom policy to configure Google+ as a claims provider in the intended user journey(s).
4. Upload the custom policy.
5. Test the custom policy using **Run Now**.

The next sections detail the above.

Creating a Google+ application

To create a Google+ application and add its details to your B2C tenant, you can follow and complete all the required instructions in the article AZURE ACTIVE DIRECTORY B2C: PROVIDE SIGN-UP AND SIGN-IN TO CONSUMERS WITH GOOGLE+ ACCOUNTS[26].

You will need a Google+ account to perform the described steps. If you don't have one, you can get it at https://accounts.google.com/SignUp?hl=en.

To illustrate the core principles that pertain to the integration of any OAuth 2.0 based identity provider, we cover hereafter the declaration of your B2C tenant as an application in this social identity provider.

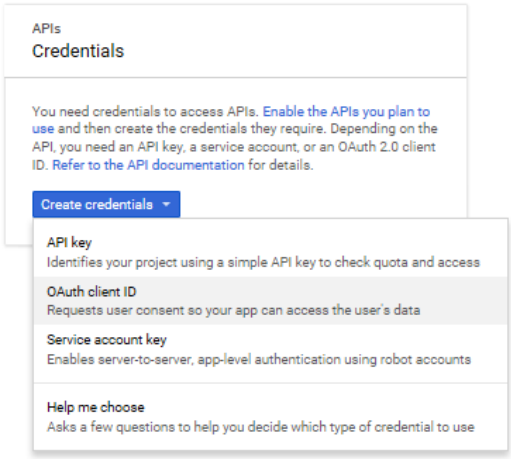
Proceed with the following steps:

1. Open a browsing session and navigate to the Google Developer Console at https://console.developers.google.com/.
2. Sign in to your Google+ account.
3. Select **Select project**, then and **Create**.
4. Enter a project name, for example "Litware369B2CGooglePlus", and then click **Create**.
5. When the project has been created, select the **Litware369B2CGooglePlus** project from project menu.

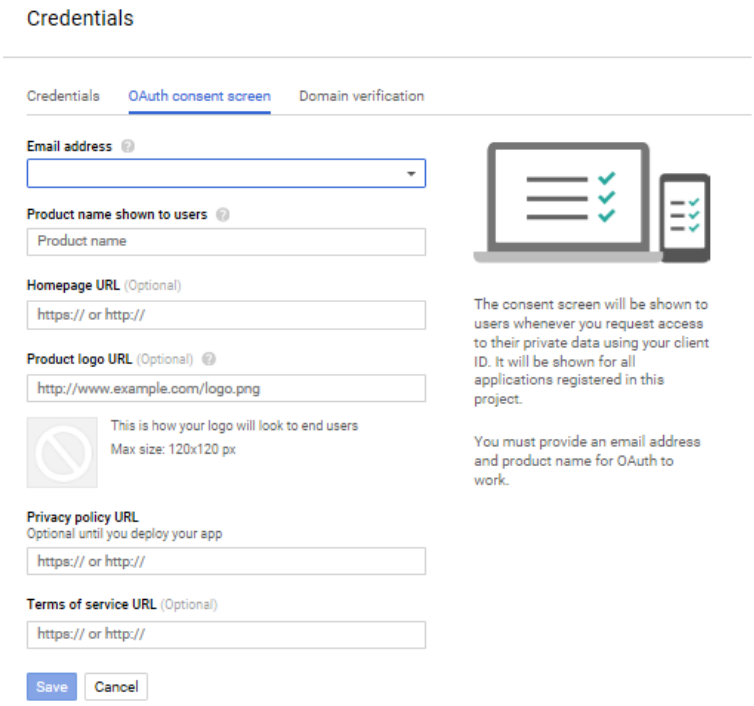
6. In the left-hand menu, select **API & Services**, and then **Credentials**.

[25] AZURE ACTIVE DIRECTORY B2C: ADD GOOGLE+ AS AN OAUTH2 IDENTITY PROVIDER USING CUSTOM POLICIES: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-custom-setup-goog-idp
[26] AZURE ACTIVE DIRECTORY B2C: PROVIDE SIGN-UP AND SIGN-IN TO CONSUMERS WITH GOOGLE+ ACCOUNTS: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-setup-goog-app

7. Select **Create credentials**, and then **OAuth client ID**.

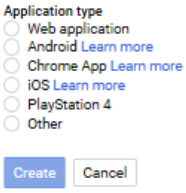


8. Click **Configure consent screen**.

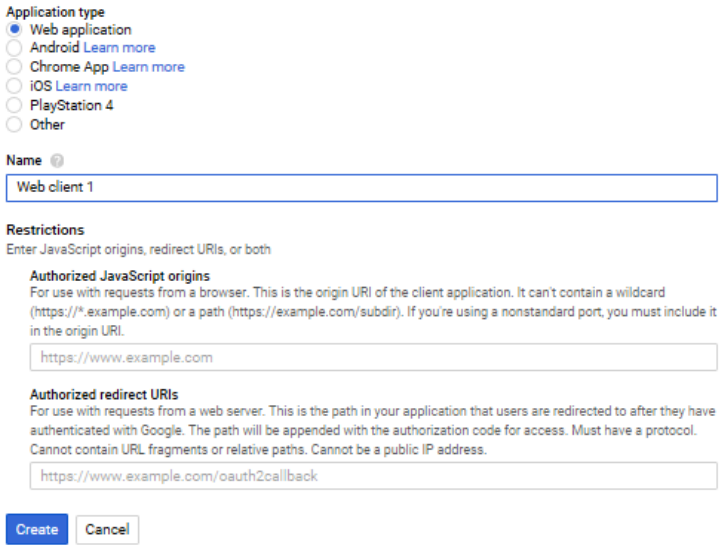


   a. in **Email address**, provide your email address,
   b. In **Product name shown to users**, enter a product name.
   c. Select **Save**.

9. A list of options shows up.



Select **Web Application.**



10. Configure the Web application for your B2C tenant:

    a. In Name, enter "*Azure AD B2C client*".
    b. In **Authorized JavaScript Origins**, enter "*https://login.microsoftonline.com*".
    c. In **Authorized Redirect URIs**, use

    https://login.microsoftonline.com/te/<*your_b2c_tenant*>.onmicrosoft.com/oauth2/authresp

    where *your_b2c_tenant* the name of your B2C tenant, for example:

    https://login.microsoftonline.com/te/litware369b2c.onmicrosoft.com/oauth2/authresp

    d. Click **Create**.

11. Record both the **client ID** and **client secret**; you will need them later to configure Google+ as a social identity provider in your B2C tenant (see next section and section § *Defining Google+ as a claim provider*). They will be respectively referred as to the "*YourAppIDValue*" and "*YourAppSecretValue*" values.

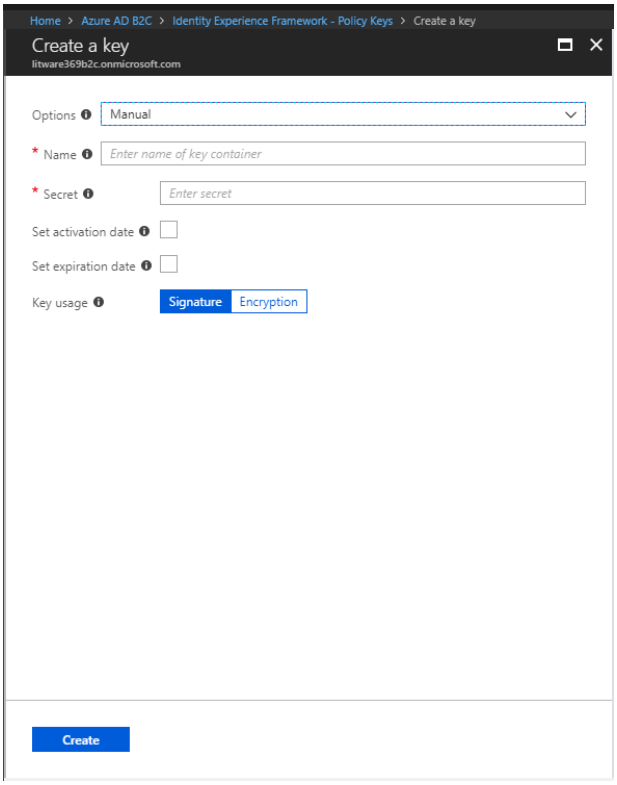
Adding the Google+ application key in your B2C tenant

Federation with Google+ accounts requires a client secret for Google+ to trust Azure AD B2C on behalf of the above application. This secret, i.e. the above "*YourAppSecretValue*" value need to be stored in your B2C tenant.

Proceed with the following steps:

11. Log in to the Azure portal as an account with administrative privileges in your Azure tenant, and then switch to your B2C tenant, for example **litware369b2.onmicrosoft.com** in our illustration.
12. Select **All services**, type "Azure AD B2C" in the search field, and then select **Azure AD B2C**.
13. Select **Identity Framework Experience - PREVIEW**.
14. Select **Policy Keys** from left menu.
15. Select **+Add**.

16. For **Options**, select **Manual**.



17. For **Key usage**, leave **Signature** selected.
18. In **Name**, enter "*GoogleSecret*".
19. In **Secret**, enter the Google app secret key you recorded earlier, i.e. the above "*YourAppSecretValue*" value.
20. Select **Create**.

Once the operation complete, a key **B2C_1A_GoogleSecret** should be created and listed.

Updating the custom policy

In lieu of the below steps, you can use the sample policy files under the folder *scenarios\aadb2c-ief-setup-google-app* of the "Starter Pack" for enabling a Google+ application as an identity provider (IdP) in your B2C tenant.

Defining Google+ as a claim provider

To sign in by using Google+ account, you need to define Google+ as a claims provider in your B2C tenant. In other words, you need to specify an endpoint that Azure AD B2C communicates with. The endpoint provides a set of claims that are used by Azure AD B2C to verify that a specific user has authenticated.

It's time to define Google+ as a claims provider with an associated technical profile. This technical profile is named **Google-OAUTH** hereafter.

Proceed with the following steps:

1. Navigate to the *SocialAndLocalAccounts* folder in the "Starter Pack".
2. Open the extension policy file, i.e. the *TrustFrameworkExtensions.xml* file, using an XML editor of your choice, for instance Visual Studio (Code).
3. Scroll down to the *ClaimsProviders* section.
4. Add the following *ClaimsProvider* element to the list in the *ClaimsProviders* node. Replace your_Google+_client_id with the client ID of your Google+ application, i.e. the above the "*YourAppIDValue*" value.

```
<!-- Google+ claims provider -->
<ClaimsProvider>
   <Domain>google.com</Domain>
   <DisplayName>Google</DisplayName>
   <TechnicalProfiles>
      <TechnicalProfile Id="Google-OAUTH">
         <DisplayName>Google</DisplayName>
         <Protocol Name="OAuth2" />
         <Metadata>
            <Item Key="ProviderName">google</Item>
            <Item Key="authorization_endpoint">https://accounts.google.com/o/oauth2/auth</Item>
            <Item Key="AccessTokenEndpoint">https://accounts.google.com/o/oauth2/token</Item>
            <Item Key="ClaimsEndpoint">https://www.googleapis.com/oauth2/v1/userinfo</Item>
            <Item Key="scope">email</Item>
            <Item Key="HttpBinding">POST</Item>
            <Item Key="UsePolicyInRedirectUri">0</Item>

            <Item Key="client_id">your_Google+_client_id</Item>

         </Metadata>
         <CryptographicKeys>
            <Key Id="client_secret" StorageReferenceId="B2C_1A_GoogleSecret" />
         </CryptographicKeys>
         <OutputClaims>
            <OutputClaim ClaimTypeReferenceId="socialIdpUserId" PartnerClaimType="id" />
            <OutputClaim ClaimTypeReferenceId="email" PartnerClaimType="email" />
            <OutputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="given_name" />
            <OutputClaim ClaimTypeReferenceId="surname" PartnerClaimType="family_name" />
            <OutputClaim ClaimTypeReferenceId="displayName" PartnerClaimType="name" />
            <OutputClaim ClaimTypeReferenceId="identityProvider" DefaultValue="google.com" />
            <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="socialIdpAuthentication" />
         </OutputClaims>
         <OutputClaimsTransformations>
            <OutputClaimsTransformation ReferenceId="CreateRandomUPNUserName" />
            <OutputClaimsTransformation ReferenceId="CreateUserPrincipalName" />
            <OutputClaimsTransformation ReferenceId="CreateAlternativeSecurityId" />
            <OutputClaimsTransformation ReferenceId="CreateSubjectClaimFromAlternativeSecurityId" />
         </OutputClaimsTransformations>
         <UseTechnicalProfileForSessionManagement ReferenceId="SM-SocialLogin" />
         <ErrorHandlers>
            <ErrorHandler>
               <ErrorResponseFormat>json</ErrorResponseFormat>
               <ResponseMatch>$[?(@.error == 'invalid_grant')]</ResponseMatch>
               <Action>Reauthenticate</Action>
               <!--In case of authroziation code used error, we don't want the user to select his account again.-->
               <!--AdditionalRequestParameters Key="prompt">select_account</AdditionalRequestParameters-->
            </ErrorHandler>
         </ErrorHandlers>
      </TechnicalProfile>
   </TechnicalProfiles>
</ClaimsProvider>
```

Note For more information, see section § *Specifying a technical profile for an OAuth 2.0 claims provider* in the sixth document of this series, along with the implementation notes for the technical profile if any.

In addition, federation with Google+ requires a client secret for Google+ to trust Azure AD B2C on behalf of the application. The above XML code snippet already references the same Google secret named **B2C_1A_GoogleSecret** that was created through the B2C blade of the Azure portal if you have followed the procedure outlined in the previous section.

5. Save the XML file.

Registering the Google+ claims provider to the Sign-Up or Sign-In (SUSI) user journey

To register the Google+ claims provider to the Sign-Up or Sign-In (SUSI) user journey, proceed with the following steps:

1. Still from the *SocialAndLocalAccounts* folder in the "Starter Pack", open the *TrustFrameworkExtensions.xml* file using an XML editor of your choice.
2. Find the *UserJourneys* element. If the element doesn't exist, add one (or uncomment it).
3. Copy the entire content of *UserJourneys* node from the *TrustFrameworkBase.xml* file as a child of the *UserJourneys* element in the *TrustFrameworkExtensions.xml* file. If you already copied this content, you can skip this step.
4. Find the *UserJourney* element with attribute value Id="SignUpOrSignIn".
5. Locate the *OrchestrationStep* element with attribute Order="1" value under this element.
6. Add the following XML snippet under the *ClaimsProviderSelections* node:

```
<ClaimsProviderSelection TargetClaimsExchangeId="GoogleExchange" />
```

7. Locate the *OrchestrationStep* element with attribute Order="2" in the *UserJourney* node and add the following XML snippet under the *ClaimExchanges* node:

```
<ClaimsExchange Id="GoogleExchange" TechnicalProfileReferenceId="Google-OAUTH" />
```

8. Save the XML file.

Registering the Google+ claims provider to the Profile Edit user journey (Optional)

To register the Google+ claims provider to the Profile Edit user journey, proceed with the following steps:

1. Still from the *SocialAndLocalAccounts* folder in the "Starter Pack", open the *TrustFrameworkExtensions.xml* file using an XML editor of your choice.
2. Find the *UserJourneys* element, and then find the *UserJourney* element with attribute value Id="ProfileEdit".
3. Locate the *OrchestrationStep* element with attribute Order="1" value under this element.
4. Add the following XML snippet under the *ClaimsProviderSelections* node:

```
<ClaimsProviderSelection TargetClaimsExchangeId="GoogleExchange" />
```

5. Locate the *OrchestrationStep* element with attribute Order="2" in the *UserJourney* node and add the following XML snippet under the *ClaimExchanges* node:

```
<ClaimsExchange Id="GoogleExchange" TechnicalProfileReferenceId="Google-OAUTH" />
```

6. Save the XML file.

Uploading the custom policy to your B2C tenant

Proceed with the following steps:

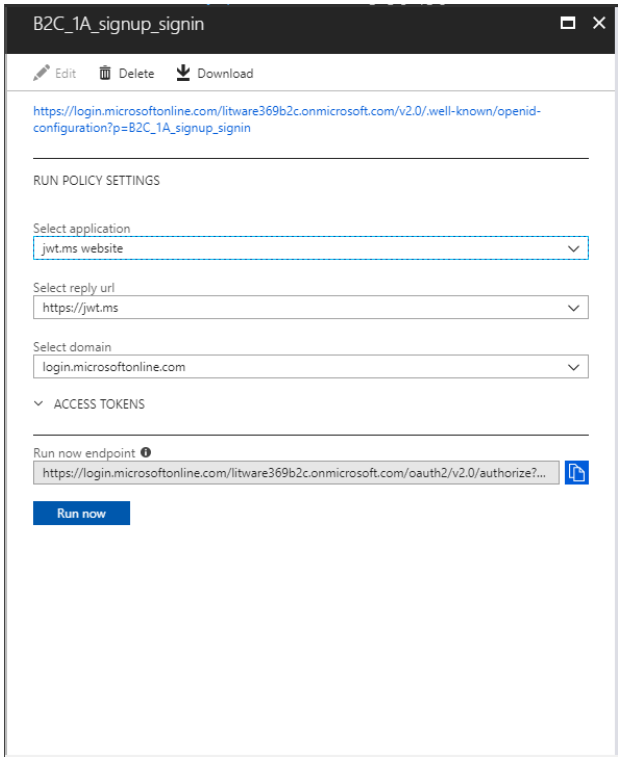
6. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
7. Select **Upload Policy**.
8. Check **Overwrite the policy if it exists**.
9. In **Upload policy**, select the *TrustFrameworkExtensions.xml* policy file
10. Click **Upload** and ensure that it does not fail the validation.

Testing the custom policy by using Run Now

Proceed with the following steps:

1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.

2. Click the **B2C_1A_signup_signin** custom policy to see its detail view. A new blade opens.



3. Click **Run now**.
4. The sign-in page should include the option to sign in with Google+. You should be able to sign in using your Google+ account.

Integrating Amazon as a claims provider

This section shows you how to configure Amazon as a claims provider in your B2C tenant.

If you're not interested in this use case, you can skip this entire section and jump to the next one.

To do so, you will have to:

1. Create an Amazon application and supply it with the right parameters.
2. Add the Amazon application key in your B2C tenant.
3. Update the custom policy to configure Amazon as a claims provider in the intended user journey(s).
4. Upload the custom policy.
5. Test the custom policy using **Run Now**.

The next sections detail the above.

Creating an Amazon application

The article AZURE ACTIVE DIRECTORY B2C: PROVIDE SIGN-UP AND SIGN-IN TO CONSUMERS WITH AMAZON ACCOUNTS[27] provides all the required instructions for creating the Amazon application:

[27] AZURE ACTIVE DIRECTORY B2C: PROVIDE SIGN-UP AND SIGN-IN TO CONSUMERS WITH AMAZON ACCOUNTS: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-setup-amzn-app

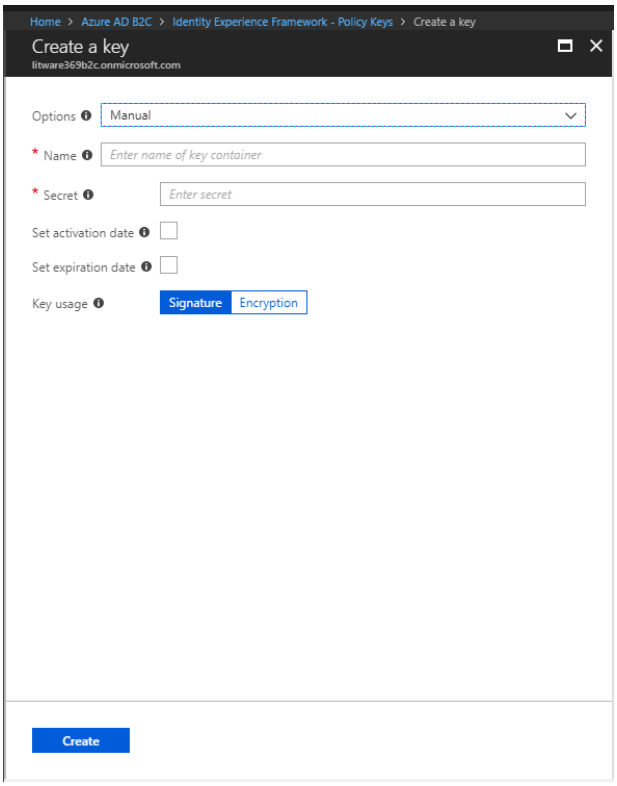
- You will need an Amazon account to perform the described steps. If you don't have one, you can get it at https://www.amazon.com/.
- Eventually, you will need the value of both **Client ID** and **Client Secret** to configure Amazon as a social identity provider in your B2C tenant. Note down these values that will be in the next section respectively referred as to the "*YourClientIDValue*" and "*YourClientSecretValue*" values.

Adding the Amazon application key in your B2C tenant

Federation with Amazon accounts requires a client secret for Amazon to trust Azure AD B2C on behalf of the above application. This secret, i.e. the above "*YourAppSecretValue*" value need to be stored in your B2C tenant.

Proceed with the following steps:

1. Log in to the Azure portal as an account with administrative privileges in your Azure tenant, and then switch to your B2C tenant, for example **litware369b2.onmicrosoft.com** in our illustration.
2. Select **All services**, type **Azure AD BC** in the search field, and then select **Azure AD B2C**.
3. Select **Identity Framework Experience - PREVIEW**.
4. Select **Policy Keys** from left menu.
5. Select **+Add**.
6. For **Options**, select **Manual**.
7. For **Key usage**, leave **Signature** selected.



8. In **Name**, enter "*AmazonSecret*".

9. In **Secret**, enter the Amazon app secret key you recorded earlier, i.e. the above "*YourAppSecretValue*" value.
10. Select **Create**.

Once the operation complete, a key **B2C_1A_AmazonSecret** should be created and listed.

Updating the custom policy

In lieu of the below steps, you can use (the content of) the sample policy file *Sample-Amazon-OAuth2-TrustFrameworkExtensions.xml* that is provided under the folder *scenarios\aadb2c-ief-setup-amzn-app* of the "Starter Pack" for enabling an Amazon application as an Identity Provider in your B2C tenant.

Defining Amazon as a claim provider

You will need at this stage to configure a claims provider for Amazon and reflect in its definition the above values.

As before, we continue to leverage the *B2C_1A_TrustFrameworkExtensions.xml* custom policy file of the SocialAndLocalAccounts core template of the "Starter Pack" to provide a core definition for Amazon as a claims provider with an associated technical profile. This technical profile is named **Amazon-OAUTH** hereafter.

```
<!-- Amazon claims provider -->
<ClaimsProvider>
   <Domain>amazon.com</Domain>
   <DisplayName>Amazon</DisplayName>
   <TechnicalProfiles>
      <TechnicalProfile Id="Amazon-OAUTH">
         <DisplayName>Amazon</DisplayName>
         <Protocol Name="OAuth2" />
         <Metadata>
            <Item Key="ProviderName">amazon</Item>
            <Item Key="authorization_endpoint">https://www.amazon.com/ap/oa</Item>
            <Item Key="AccessTokenEndpoint">https://api.amazon.com/auth/o2/token</Item>
            <Item Key="ClaimsEndpoint">https://api.amazon.com/user/profile</Item>
            <Item Key="scope">profile</Item>
            <Item Key="HttpBinding">POST</Item>
            <Item Key="UsePolicyInRedirectUri">0</Item>

            <Item Key="client_id">your_Amazon_client_id</Item>

         </Metadata>
         <CryptographicKeys>
            <Key Id="client_secret" StorageReferenceId="B2C_1A_AmazonSecret" />
         </CryptographicKeys>
         <OutputClaims>
            <OutputClaim ClaimTypeReferenceId="socialIdpUserId" PartnerClaimType="user_id" />
            <OutputClaim ClaimTypeReferenceId="email" PartnerClaimType="email" />
            <OutputClaim ClaimTypeReferenceId="displayName" PartnerClaimType="name" />
            <OutputClaim ClaimTypeReferenceId="identityProvider" DefaultValue="amazon.com" />
            <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="socialIdpAuthentication" />
         </OutputClaims>
         <OutputClaimsTransformations>
            <OutputClaimsTransformation ReferenceId="CreateRandomUPNUserName" />
            <OutputClaimsTransformation ReferenceId="CreateUserPrincipalName" />
            <OutputClaimsTransformation ReferenceId="CreateAlternativeSecurityId" />
            <OutputClaimsTransformation ReferenceId="CreateSubjectClaimFromAlternativeSecurityId" />
         </OutputClaimsTransformations>
         <UseTechnicalProfileForSessionManagement ReferenceId="SM-SocialLogin" />
      </TechnicalProfile>
   </TechnicalProfiles>
</ClaimsProvider>
```

Note For more information, see section § *Specifying a technical profile for an OAuth 2.0 claims provider* in the sixth document of this series, along with the implementation notes for the technical profile if any.

You simply need to slightly modify the above XML code snippet to reflect in it the information that pertains to the Amazon application created in the previous section. Replace your_Amazon_client_id with the client ID of your Amazon application, i.e. the above the "*YourAppIDValue*" value.

In addition, federation with Amazon requires a client secret for Amazon to trust Azure AD B2C on behalf of the application. The above XML code snippet already references the same Amazon secret named **B2C_1A_AmazonSecret** that was created through the B2C blade of the Azure portal if you have followed the procedure outlined in the previous section.

Registering the Amazon claims provider to the Sign-Up or Sign-In (SUSI) user journey

At this stage, Amazon has been set up as a claims provider but it's not available in any user journey. To make it available, we need to create a duplicate of an existing user journey template and to add the Amazon claims provider in it.

To register the Amazon claims provider to the Sign-Up or Sign-In (SUSI) user journey, proceed with the following steps:

1. Still from the *SocialAndLocalAccounts* folder in the "Starter Pack", open the *TrustFrameworkExtensions.xml* file using an XML editor of your choice.
2. Find the *UserJourneys* element. If the element doesn't exist, add one (or uncomment it).
3. Copy the entire content of *UserJourneys* node from the *TrustFrameworkBase.xml* file as a child of the *UserJourneys* element in the *TrustFrameworkExtensions.xml* file. If you already copied this content, you can skip this step.
4. Find the *UserJourney* element with attribute value Id="SignUpOrSignIn".
5. Locate the *OrchestrationStep* element with attribute Order="1" value under this element.
6. Add the following XML snippet under the *ClaimsProviderSelections* node:

```
<ClaimsProviderSelection TargetClaimsExchangeId="AmazonExchange" />
```

7. Locate the *OrchestrationStep* element with attribute Order="2" in the *UserJourney* node and add the following XML snippet under the *ClaimExchanges* node:

```
<ClaimsExchange Id="AmazonExchange" TechnicalProfileReferenceId="Amazon-OAUTH" />
```

8. Save the XML file.

Registering the Amazon claims provider to the Profile Edit user journey (Optional)

To register the Amazon claims provider to the Profile Edit user journey, proceed with the following steps:

1. Still from the *SocialAndLocalAccounts* folder in the "Starter Pack", open the *TrustFrameworkExtensions.xml* file using an XML editor of your choice.
2. Find the *UserJourneys* element, and then find the *UserJourney* element with attribute value Id="ProfileEdit".
3. Locate the *OrchestrationStep* element with attribute Order="1" value under this element.
4. Add the following XML snippet under the *ClaimsProviderSelections* node:

```
<ClaimsProviderSelection TargetClaimsExchangeId="AmazonExchange" />
```

5. Locate the *OrchestrationStep* element with attribute Order="2" in the *UserJourney* node and add the following XML snippet under the *ClaimExchanges* node:

```
<ClaimsExchange Id="AmazonExchange" TechnicalProfileReferenceId="Amazon-OAUTH" />
```

6. Save the XML file.

Uploading the custom policy to your B2C tenant

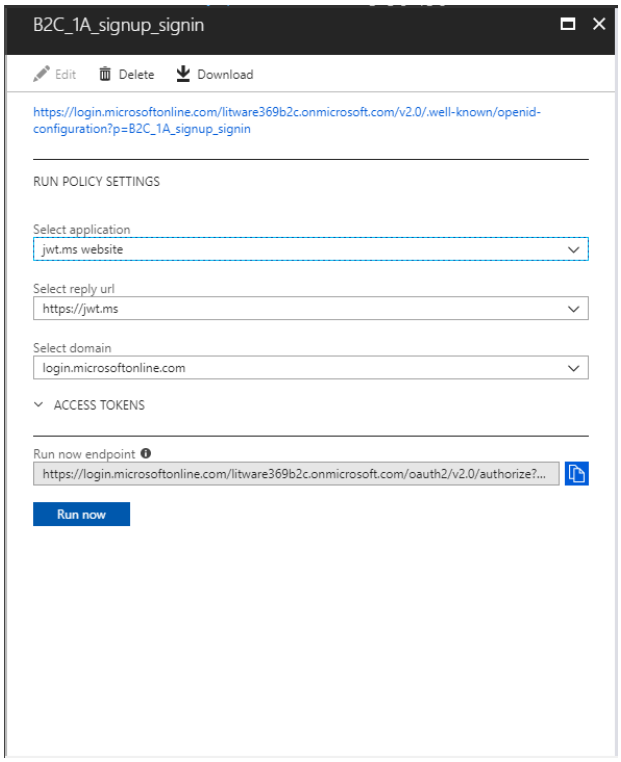
Proceed with the following steps:

1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
2. Select **Upload Policy**.
3. Check **Overwrite the policy if it exists**.
4. In **Upload policy**, select the *TrustFrameworkExtensions.xml* policy file
5. Click **Upload** and ensure that it does not fail the validation.

Testing the custom policy by using Run Now

Proceed with the following steps:

1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
2. Click the **B2C_1A_signup_signin** custom policy to see its detail view. (You can alternatively select the **B2C_1A_ProfileEdit** custom policy if you also modified above the Profile Edit user journey.) A new blade opens.



3. Click **Run now**.

The sign-in page should include the option to sign in with Amazon. You should be able to sign in using your Amazon account.

Integrating LinkedIn as a claims provider

This section shows you, in this use case, how to configure LinkedIn as a social identity provider in your B2C tenant.

> **Note** For more information, see article AZURE ACTIVE DIRECTORY B2C: ADD LINKEDIN AS AN IDENTITY PROVIDER BY USING CUSTOM POLICIES[28].

If you're not interested in this use case, you can skip this entire section and jump to the next one.

To do so, you will have to:

1. Create a LinkedIn application and supply it with the right parameters.
2. Add the LinkedIn application key in your B2C tenant.
3. Update the custom policy to configure LinkedIn as a claims provider in the intended user journey(s).
4. Upload the custom policy.
5. Test the custom policy using **Run Now**.

The next sections detail the above.

Creating a LinkedIn application

The article AZURE ACTIVE DIRECTORY B2C: PROVIDE SIGN-UP AND SIGN-IN TO CONSUMERS WITH LINKEDIN ACCOUNTS[29] provides all the required instructions for creating the LinkedIn application:

- You will need a LinkedIn account to perform the described steps. If you don't have one, you can get it at https://www.linkedin.com/.
- Eventually, you will need the value of both **Client ID** and **Client Secret** to configure LinkedIn as a social identity provider in your B2C tenant. Note down these values that will be later respectively referred as to the "*YourClientIDValue*" and "*YourClientSecretValue*" values.

Adding the LinkedIn application key in your B2C tenant

Federation with LinkedIn accounts requires a client secret for LinkedIn to trust Azure AD B2C on behalf of the above application. This secret, i.e. the above "*YourAppSecretValue*" value need to be stored in your B2C tenant.

Proceed with the following steps:

1. Log in to the Azure portal as an account with administrative privileges in your Azure tenant, and then switch to your B2C tenant, for example **litware369b2.onmicrosoft.com** in our illustration.

---

[28] AZURE ACTIVE DIRECTORY B2C: ADD LINKEDIN AS AN IDENTITY PROVIDER BY USING CUSTOM POLICIES: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-custom-setup-li-idp

[29] AZURE ACTIVE DIRECTORY B2C: PROVIDE SIGN-UP AND SIGN-IN TO CONSUMERS WITH LINKEDIN ACCOUNTS: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-setup-li-app

2. Select **All services**, type **Azure AD BC** in the search field, and then select **Azure AD B2C**.
3. Select **Identity Framework Experience - PREVIEW**.
4. Select **Policy Keys** from left menu.
5. Select **+Add**.
6. For **Options**, select **Manual**.
7. For **Key usage**, select **Signature**.
8. In **Name**, enter "*LinkedInSecret*".
9. In **Secret**, enter the LinkedIn app secret key you recorded earlier, i.e. the above "*YourAppSecretValue*" value.
10. Select **Create**.

Once the operation complete, a key **B2C_1A_LinkedInSecret** should be created and listed.

Updating the custom policy

Based on the outcome of the previous section, the configuration for LinkedIn as a claims provider in your custom policy is straightforward as a simple two steps process. This twofold process can be described as follows:

1. Add LinkedIn as a claims provider in the *B2C_1A_TrustFrameworkExtensions.xml* custom policy file that comes along with the core templates of the "Starter Pack".
2. Use LinkedIn as a claims provider in an (existing) user journey.

The next sections detail the above.

In lieu of the below steps, you can use the sample policy file *Sample-LinkdIn-OAuth2-TrustFrameworkExtensions.xml* under the folder *scenarios\aadb2c-ief-setup-li-app* of the "Starter Pack" for enabling a LinkedIn application as an identity provider (IdP) in your B2C tenant.

Defining LinkedIn as a claim provider

You will need at this stage to configure a claims provider for LinkedIn and reflect in its definition the above values.

As before, we continue to leverage the *B2C_1A_TrustFrameworkExtensions.xml* custom policy file of the SocialAndLocalAccounts core template of the "Starter Pack" to provide a core definition for LinkedIn as a claims provider with an associated technical profile. This technical profile is named **LinkedIn-OAUTH** hereafter.

```
<!-- LinkedIn claims provider -->
<ClaimsProvider>
   <Domain>linkedin.com</Domain>
   <DisplayName>LinkedIn</DisplayName>
   <TechnicalProfiles>
      <TechnicalProfile Id="LinkedIn-OAUTH">
         <DisplayName>LinkedIn</DisplayName>
         <Protocol Name="OAuth2" />
         <Metadata>
            <Item Key="ProviderName">linkedin</Item>
            <Item Key="authorization_endpoint">https://www.linkedin.com/oauth/v2/authorization</Item>
            <Item Key="AccessTokenEndpoint">https://www.linkedin.com/oauth/v2/accessToken</Item>
            <Item Key="ClaimsEndpoint">https://api.linkedin.com/v1/people/~:(id,first-name,last-name,email-
address,headline)</Item>
```

```
            <Item Key="ClaimsEndpointAccessTokenName">oauth2_access_token</Item>
            <Item Key="ClaimsEndpointFormatName">format</Item>
            <Item Key="ClaimsEndpointFormat">json</Item>
            <Item Key="scope">r_emailaddress r_basicprofile</Item>
            <Item Key="HttpBinding">POST</Item>
            <Item Key="UsePolicyInRedirectUri">0</Item>

            <Item Key="client_id">your_LinkedIn_client_id</Item>

         </Metadata>
         <CryptographicKeys>
            <Key Id="client_secret" StorageReferenceId="B2C_1A_LinkedInSecret" />
         </CryptographicKeys>
         <OutputClaims>
            <OutputClaim ClaimTypeReferenceId="socialIdpUserId" PartnerClaimType="id" />
            <OutputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="firstName" />
            <OutputClaim ClaimTypeReferenceId="surname" PartnerClaimType="lastName" />
            <OutputClaim ClaimTypeReferenceId="email" PartnerClaimType="emailAddress" />
            <!--<OutputClaim ClaimTypeReferenceId="jobTitle" PartnerClaimType="headline" />-->
            <OutputClaim ClaimTypeReferenceId="identityProvider" DefaultValue="linkedin.com" />
            <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="socialIdpAuthentication" />
         </OutputClaims>
         <OutputClaimsTransformations>
            <OutputClaimsTransformation ReferenceId="CreateRandomUPNUserName" />
            <OutputClaimsTransformation ReferenceId="CreateUserPrincipalName" />
            <OutputClaimsTransformation ReferenceId="CreateAlternativeSecurityId" />
            <OutputClaimsTransformation ReferenceId="CreateSubjectClaimFromAlternativeSecurityId" />
         </OutputClaimsTransformations>
         <UseTechnicalProfileForSessionManagement ReferenceId="SM-SocialLogin" />
      </TechnicalProfile>
   </TechnicalProfiles>
</ClaimsProvider>
```

Note For more information, see section § *Specifying a technical profile for an OAuth 2.0 claims provider* in the sixth document of this series, along with the implementation notes for the technical profile if any.

You simply need to slightly modify the above XML code snippet to reflect in it the information that pertains to the LinkedIn application created in the previous section. Replace your_LinkedIn_client_id with the client ID of your LinkedIn application, i.e. the above the "*YourAppIDValue*" value.

In addition, federation with LinkedIn requires a client secret for LinkedIn to trust Azure AD B2C on behalf of the application. The above XML code snippet already references the same LinkedIn secret named **B2C_1A_LinkedIn Secret** that was created through the B2C blade of the Azure portal if you have followed the procedure outlined in the previous section.

Registering the LinkedIn claims provider to the Sign-Up or Sign-In (SUSI) user journey

To register the LinkedIn claims provider to the Sign-Up or Sign-In (SUSI) user journey, proceed with the following steps:

1. Still from the *SocialAndLocalAccounts* folder in the "Starter Pack", open the *TrustFrameworkExtensions.xml* file using an XML editor of your choice.
2. Find the *UserJourneys* element. If the element doesn't exist, add one (or uncomment it).
3. Copy the entire content of *UserJourneys* node from the *TrustFrameworkBase.xml* file as a child of the *UserJourneys* element in the *TrustFrameworkExtensions.xml* file. If you already copied this content, you can skip this step.
4. Find the *UserJourney* element with attribute value Id="SignUpOrSignIn".
5. Locate the *OrchestrationStep* element with attribute Order="1" value under this element.

6. Add the following XML snippet under the *ClaimsProviderSelections* node:

```
<ClaimsProviderSelection TargetClaimsExchangeId="LinkedInExchange" />
```

7. Locate the *OrchestrationStep* element with attribute Order="2" in the *UserJourney* node and add the following XML snippet under the *ClaimExchanges* node:

```
<ClaimsExchange Id="LinkedInExchange" TechnicalProfileReferenceId="LinkedIn-OAUTH" />
```

8. Save the XML file.

Registering the LinkedIn claims provider to the Profile Edit user journey (Optional)

To register the LinkedIn claims provider to the Profile Edit user journey, proceed with the following steps:

1. Still from the *SocialAndLocalAccounts* folder in the "Starter Pack", open the *TrustFrameworkExtensions.xml* file using an XML editor of your choice.
2. Find the *UserJourneys* element, and then find the *UserJourney* element with attribute value Id="ProfileEdit".
3. Locate the *OrchestrationStep* element with attribute Order="1" value under this element.
4. Add the following XML snippet under the *ClaimsProviderSelections* node:

```
<ClaimsProviderSelection TargetClaimsExchangeId="LinkedInExchange" />
```

5. Locate the *OrchestrationStep* element with attribute Order="2" in the *UserJourney* node and add the following XML snippet under the *ClaimExchanges* node:

```
<ClaimsExchange Id="LinkedInExchange" TechnicalProfileReferenceId="LinkedIn-OAUTH" />
```

6. Save the XML file.

Uploading the custom policy to your B2C tenant

Proceed with the following steps:

1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
2. Select **Upload Policy**.
3. Check **Overwrite the policy if it exists**.
4. In **Upload policy**, select the *TrustFrameworkExtensions.xml* policy file
5. Click **Upload** and ensure that it does not fail the validation.

Testing the custom policy by using Run Now

Proceed with the following steps:

1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
2. Click the **B2C_1A_signup_signin** custom policy to see its detail view. (You can alternatively select the **B2C_1A_ProfileEdit custom** policy if you also modified above the Profile Edit user journey.) A new blade opens.
3. Click **Run now**.
4. The sign-in page should include the option to sign in with LinkedIn. You should be able to sign in using your LinkedIn account.

Integrating Microsoft Account (MSA) as a claims provider

This section shows you, in this use case, how to configure Microsoft Account (MSA) as a social identity provider in your B2C tenant.

> **Note** For more information, see article AZURE ACTIVE DIRECTORY B2C: ADD MICROSOFT ACCOUNT (MSA) AS AN IDENTITY PROVIDER USING CUSTOM POLICIES[30].

If you're not interested in this use case, you can skip this entire section and jump to the next one.

To do so, you will have to:

1. Create a Microsoft Account application and supply it with the right parameters.
2. Add the Microsoft Account application key in your B2C tenant.
3. Update the custom policy to configure Microsoft Account as a claims provider in the intended user journey(s).
4. Upload the custom policy.
5. Test the custom policy using **Run Now**.

The next sections detail the above.

Creating a Microsoft Account application

The article AZURE ACTIVE DIRECTORY B2C: PROVIDE SIGN-UP AND SIGN-IN TO CONSUMERS WITH MICROSOFT ACCOUNTS[31] provides all the required instructions for creating the Microsoft Account application:

- You will need a Microsoft Account (MSA) account to perform the described steps. If you don't have one, you can get it at https://www.live.com/.
- Eventually, you will need the value of both **Client ID** and **Client Secret** to configure Microsoft Account as a social identity provider in your B2C tenant. Note down these values that will be later respectively referred as to the "*YourClientIDValue*" and "*YourClientSecretValue*" values.

---

[30] AZURE ACTIVE DIRECTORY B2C: ADD MICROSOFT ACCOUNT (MSA) AS AN IDENTITY PROVIDER USING CUSTOM POLICIES: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-custom-setup-msa-idp

[31] AZURE ACTIVE DIRECTORY B2C: PROVIDE SIGN-UP AND SIGN-IN TO CONSUMERS WITH MICROSOFT ACCOUNTS: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-setup-msa-app

Adding the Microsoft Account application key in your B2C tenant

Federation with Microsoft Account (MSA) accounts requires a client secret for Microsoft Account to trust Azure AD B2C on behalf of the above application. This secret, i.e. the above "*YourAppSecretValue*" value need to be stored in your B2C tenant.

Proceed with the following steps:

1. Log in to the Azure portal as an account with administrative privileges in your Azure tenant, and then switch to your B2C tenant, for example **litware369b2.onmicrosoft.com** in our illustration.
2. Select **All services**, type **Azure AD BC** in the search field, and then select **Azure AD B2C**.
3. Select **Identity Framework Experience - PREVIEW**.
4. Select **Policy Keys** from left menu.
5. Select **+Add**.
6. For **Options**, select **Manual**.
7. For **Key usage**, select **Signature**.
8. In **Name**, enter "*MSASecret*".
9. In **Secret**, enter the Microsoft Account app secret key you recorded earlier, i.e. the above "*YourAppSecretValue*" value.
10. Select **Create**.

Once the operation complete, a key **B2C_1A_MSASecret** should be created and listed.

Updating the custom policy

Based on the outcome of the previous section, the configuration for Microsoft Account (MSA) as a claims provider in your custom policy is straightforward as a simple two steps process. This twofold process can be described as follows:

1. Add Microsoft Account as a claims provider in the *B2C_1A_TrustFrameworkExtensions.xml* custom policy file that comes along with the core templates of the "Starter Pack".
2. Use Microsoft Account as a claims provider in an (existing) user journey.

The next sections detail the above.

In lieu of the below steps, you can use the sample policy files under the folder *scenarios\aadb2c-ief-setup-msa-app* of the "Starter Pack" for enabling a Microsoft Account application as an identity provider (IdP) in your B2C tenant.

Defining Microsoft Account as a claim provider

You will need at this stage to configure a claims provider for Microsoft Account and reflect in its definition the above values.

As before, we continue to leverage the *B2C_1A_TrustFrameworkExtensions.xml* custom policy file of the SocialAndLocalAccounts core template of the "Starter Pack" to provide a core definition for Microsoft

Account as a claims provider with an associated technical profile. This technical profile is named **MSA-OIDC** hereafter.

```
<!—Microsoft Account claims provider -->
<ClaimsProvider>
   <Domain>live.com</Domain>
   <DisplayName>Microsoft Account</DisplayName>
   <TechnicalProfiles>
      <TechnicalProfile Id="MSA-OIDC">
         <DisplayName>Microsoft Account</DisplayName>
         <Protocol Name="OpenIdConnect" />
         <Metadata>
            <Item Key="ProviderName">https://login.live.com</Item>
            <Item Key="METADATA">https://login.live.com/.well-known/openid-configuration</Item>
            <Item Key="response_types">code</Item>
            <Item Key="response_mode">form_post</Item>
            <Item Key="scope">openid profile email</Item>
            <Item Key="HttpBinding">POST</Item>
            <Item Key="UsePolicyInRedirectUri">0</Item>

            <Item Key="client_id">your_MSA_client_id</Item>

         </Metadata>
         <CryptographicKeys>
            <Key Id="client_secret" StorageReferenceId="B2C_1A_MSASecret" />
         </CryptographicKeys>
         <OutputClaims>
            <OutputClaim ClaimTypeReferenceId="identityProvider" DefaultValue="live.com" />
            <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="socialIdpAuthentication" />
            <OutputClaim ClaimTypeReferenceId="socialIdpUserId" PartnerClaimType="sub" />
            <OutputClaim ClaimTypeReferenceId="displayName" PartnerClaimType="name" />
            <OutputClaim ClaimTypeReferenceId="email" />
         </OutputClaims>
         <OutputClaimsTransformations>
            <OutputClaimsTransformation ReferenceId="CreateRandomUPNUserName" />
            <OutputClaimsTransformation ReferenceId="CreateUserPrincipalName" />
            <OutputClaimsTransformation ReferenceId="CreateAlternativeSecurityId" />
            <OutputClaimsTransformation ReferenceId="CreateSubjectClaimFromAlternativeSecurityId" />
         </OutputClaimsTransformations>
         <UseTechnicalProfileForSessionManagement ReferenceId="SM-SocialLogin" />
      </TechnicalProfile>
   </TechnicalProfiles>
</ClaimsProvider>
```

> **Note** For more information, see section § *Specifying a technical profile for an OpenID Connect claims provider* in the sixth document of this series, along with the implementation notes for the technical profile if any.

You simply need to slightly modify the above XML code snippet to reflect in it the information that pertains to the Microsoft Account application created in the previous section. Replace your_MSA_client_id with the client ID of your Microsoft Account application, i.e. the above the "*YourAppIDValue*" value.

In addition, federation with Microsoft Account requires a client secret for Microsoft Account to trust Azure AD B2C on behalf of the application. The above XML code snippet already references the same Microsoft Account secret named **B2C_1A_MSASecret** that was created through the B2C blade of the Azure portal if you have followed the procedure outlined in the previous section.

Registering the Microsoft Account claims provider to the Sign-Up or Sign-In (SUSI) user journey

To register the Microsoft Account claims provider to the Sign-Up or Sign-In (SUSI) user journey, proceed with the following steps:

1. Still from the *SocialAndLocalAccounts* folder in the "Starter Pack", open the *TrustFrameworkExtensions.xml* file using an XML editor of your choice.
2. Find the *UserJourneys* element. If the element doesn't exist, add one (or uncomment it).
3. Copy the entire content of *UserJourneys* node from the *TrustFrameworkBase.xml* file as a child of the *UserJourneys* element in the *TrustFrameworkExtensions.xml* file. If you already copied this content, you can skip this step.
4. Find the *UserJourney* element with attribute value Id="SignUpOrSignIn".
5. Locate the *OrchestrationStep* element with attribute Order="1" value under this element.
6. Add the following XML snippet under the *ClaimsProviderSelections* node:

```
<ClaimsProviderSelection TargetClaimsExchangeId="MSAExchange" />
```

7. Locate the *OrchestrationStep* element with attribute Order="2" in the *UserJourney* node and add the following XML snippet under the *ClaimExchanges* node:

```
<ClaimsExchange Id="MSAExchange" TechnicalProfileReferenceId="MSA-OIDC" />
```

8. Save the XML file.

Registering the Microsoft Account claims provider to the Profile Edit user journey (Optional)

To register the Microsoft Account claims provider to the Profile Edit user journey, proceed with the following steps:

1. Still from the *SocialAndLocalAccounts* folder in the "Starter Pack", open the *TrustFrameworkExtensions.xml* file using an XML editor of your choice.
2. Find the *UserJourneys* element, and then find the *UserJourney* element with attribute value Id="ProfileEdit".
3. Locate the *OrchestrationStep* element with attribute Order="1" value under this element.
4. Add the following XML snippet under the *ClaimsProviderSelections* node:

```
<ClaimsProviderSelection TargetClaimsExchangeId="MSAExchange" />
```

5. Locate the *OrchestrationStep* element with attribute Order="2" in the *UserJourney* node and add the following XML snippet under the *ClaimExchanges* node:

```
<ClaimsExchange Id="MSAExchange" TechnicalProfileReferenceId="MSA-OIDC" />
```

6. Save the XML file.

Uploading the custom policy to your B2C tenant

Proceed with the following steps:

1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
2. Select **Upload Policy**.
3. Check **Overwrite the policy if it exists**.
4. In **Upload policy**, select the *TrustFrameworkExtensions.xml* policy file
5. Click **Upload** and ensure that it does not fail the validation.

Testing the custom policy by using Run Now

Proceed with the following steps:

1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
2. Click the **B2C_1A_signup_signin** custom policy to see its detail view. (You can alternatively select the **B2C_1A_ProfileEdit custom** policy if you also modified above the Profile Edit user journey.) A new blade opens.
3. Click **Run now**.
4. The sign-in page should include the option to sign in with Microsoft Account. You should be able to sign in using your Microsoft Account (MSA) account.

Integrating Twitter as a claims provider

This section shows you, in this use case, how to configure Twitter as a social identity provider in your B2C tenant.

> **Note** For more information, see article AZURE ACTIVE DIRECTORY B2C: ADD TWITTER AS AN OAUTH1 IDENTITY PROVIDER BY USING CUSTOM POLICIES[32].

If you're not interested in this use case, you can skip this entire section and jump to the next one.

To do so, you will have to:

1. Create a Twitter application and supply it with the right parameters.
2. Add the Twitter application key in your B2C tenant.
3. Update the custom policy to configure Twitter as a claims provider in the intended user journey(s).
4. Upload the custom policy.
5. Test the custom policy using **Run Now**.

The next sections detail the above.

---

[32] AZURE ACTIVE DIRECTORY B2C: ADD TWITTER AS AN OAUTH1 IDENTITY PROVIDER BY USING CUSTOM POLICIES: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-custom-setup-twitter-idp

Creating a Twitter application

The article AZURE ACTIVE DIRECTORY B2C: PROVIDE SIGN-UP AND SIGN-IN TO CONSUMERS WITH TWITTER ACCOUNTS USING AZURE AD B2C[33] provides all the required instructions for creating the Twitter application:

- You will need a Twitter account to perform the described steps. If you don't have one, you can get it at https://twitter.com/signup.
- Eventually, you will need the value of both **Client ID** and **Client Secret** to configure Twitter as a social identity provider in your B2C tenant. Note down these values that will be later respectively referred as to the "*YourClientIDValue*" and "*YourClientSecretValue*" values.

Adding the Twitter application key in your B2C tenant

Federation with Twitter accounts requires a client secret for Twitter to trust Azure AD B2C on behalf of the above application. This secret, i.e. the above "*YourAppSecretValue*" value need to be stored in your B2C tenant.

Proceed with the following steps:

1. Log in to the Azure portal as an account with administrative privileges in your Azure tenant, and then switch to your B2C tenant, for example **litware369b2c.onmicrosoft.com** in our illustration.
2. Select **All services**, type **Azure AD BC** in the search field, and then select **Azure AD B2C**.
3. Select **Identity Framework Experience - PREVIEW**.
4. Select **Policy Keys** from left menu.
5. Select **+Add**.
6. For **Options**, select **Manual**.
7. For **Key usage**, select **Signature**.
8. In **Name**, enter "*TwitterSecret*".
9. In **Secret**, enter the Twitter application secret key you recorded earlier, i.e. the above "*YourAppSecretValue*" value.
10. Select **Create**.

Once the operation complete, a key **B2C_1A_TwitterSecret** should be created and listed.

Updating the custom policy

Based on the outcome of the previous section, the configuration for Twitter as a claims provider in your custom policy is straightforward as a simple two steps process. This twofold process can be described as follows:

1. Add Twitter as a claims provider in the *B2C_1A_TrustFrameworkExtensions.xml* custom policy file that comes along with the core templates of the "Starter Pack".

[33] AZURE ACTIVE DIRECTORY B2C: PROVIDE SIGN-UP AND SIGN-IN TO CONSUMERS WITH TWITTER ACCOUNTS USING AZURE AD B2C: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-setup-twitter-app

2. Use Twitter as a claims provider in an (existing) user journey.

The next sections detail the above.

In lieu of the below steps, you can use the sample policy file *Sample-Twitter-OAuth1-TrustFrameworkExtensions.xml* under the folder *scenarios\aadb2c-ief-setup-twitter-app* of the "Starter Pack" for enabling a Twitter application as an identity provider (IdP) in your B2C tenant.

Defining Twitter as a claim provider

You will need at this stage to configure a claims provider for Twitter and reflect in its definition the above values.

As before, we continue to leverage the *B2C_1A_TrustFrameworkExtensions.xml* custom policy file of the SocialAndLocalAccounts core template of the "Starter Pack" to provide a core definition for Twitter as a claims provider with an associated technical profile. This technical profile is named **Twitter-OAUTH1** hereafter.

```
<!-- Twitter claims provider -->
<ClaimsProvider>
   <Domain>twitter.com</Domain>
   <DisplayName>Twitter</DisplayName>
   <TechnicalProfiles>
      <TechnicalProfile Id="Twitter-OAUTH1">
         <DisplayName>Twitter</DisplayName>
         <Protocol Name="OAuth1" />
         <Metadata>
            <Item Key="ProviderName">Twitter</Item>
            <Item Key="authorization_endpoint">https://api.twitter.com/oauth/authenticate</Item>
            <Item Key="access_token_endpoint">https://api.twitter.com/oauth/access_token</Item>
            <Item Key="request_token_endpoint">https://api.twitter.com/oauth/request_token</Item>
            <Item
Key="ClaimsEndpoint">https://api.twitter.com/1.1/account/verify_credentials.json?include_email=true</Item>
            <Item Key="ClaimsResponseFormat">json</Item>

            <Item Key="client_id">your Twitter consumer key</Item>

         </Metadata>
         <CryptographicKeys>
            <Key Id="client_secret" StorageReferenceId="B2C_1A_TwitterSecret" />
         </CryptographicKeys>
         <InputClaims />
         <OutputClaims>
            <OutputClaim ClaimTypeReferenceId="socialIdpUserId" PartnerClaimType="user_id" />
            <OutputClaim ClaimTypeReferenceId="displayName" PartnerClaimType="screen_name" />
            <OutputClaim ClaimTypeReferenceId="email" />
            <OutputClaim ClaimTypeReferenceId="identityProvider" DefaultValue="twitter.com" />
            <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="socialIdpAuthentication" />
         </OutputClaims>
         <OutputClaimsTransformations>
            <OutputClaimsTransformation ReferenceId="CreateRandomUPNUserName" />
            <OutputClaimsTransformation ReferenceId="CreateUserPrincipalName" />
            <OutputClaimsTransformation ReferenceId="CreateAlternativeSecurityId" />
            <OutputClaimsTransformation ReferenceId="CreateSubjectClaimFromAlternativeSecurityId" />
         </OutputClaimsTransformations>
         <UseTechnicalProfileForSessionManagement ReferenceId="SM-SocialLogin" />
      </TechnicalProfile>
   </TechnicalProfiles>
</ClaimsProvider>
```

Note For more information, see section § *Specifying a technical profile for an OAuth 1.0 claims provider* in the sixth document of this series, along with the implementation notes for the technical profile if any.

You simply need to slightly modify the above XML code snippet to reflect in it the information that pertains to the Twitter application created in the previous section. Replace your_Twitter_client_id with the client ID of your Twitter application, i.e. the above the "*YourAppIDValue*" value.

In addition, federation with Twitter requires a client secret for Twitter to trust Azure AD B2C on behalf of the application. The above XML code snippet already references the same Twitter secret named **B2C_1A_TwitterSecret** that was created through the B2C blade of the Azure portal if you have followed the procedure outlined in the previous section.

Registering the Twitter claims provider to the Sign-Up or Sign-In (SUSI) user journey

To register the Twitter claims provider to the Sign-Up or Sign-In (SUSI) user journey, proceed with the following steps:

1. Still from the *SocialAndLocalAccounts* folder in the "Starter Pack", open the *TrustFrameworkExtensions.xml* file using an XML editor of your choice.
2. Find the *UserJourneys* element. If the element doesn't exist, add one (or uncomment it).
3. Copy the entire content of *UserJourneys* node from the *TrustFrameworkBase.xml* file as a child of the *UserJourneys* element in the *TrustFrameworkExtensions.xml* file. If you already copied this content, you can skip this step.
4. Find the *UserJourney* element with attribute value Id="SignUpOrSignIn".
5. Locate the *OrchestrationStep* element with attribute Order="1" value under this element.
6. Add the following XML snippet under the *ClaimsProviderSelections* node:

```
<ClaimsProviderSelection TargetClaimsExchangeId="TwitterExchange" />
```

7. Locate the *OrchestrationStep* element with attribute Order="2" in the *UserJourney* node and add the following XML snippet under the *ClaimExchanges* node:

```
<ClaimsExchange Id="TwitterExchange" TechnicalProfileReferenceId="Twitter-OAUTH1" />
```

8. Save the XML file.

Registering the Twitter claims provider to the Profile Edit user journey (Optional)

To register the Twitter claims provider to the Profile Edit user journey, proceed with the following steps:

1. Still from the *SocialAndLocalAccounts* folder in the "Starter Pack", open the *TrustFrameworkExtensions.xml* file using an XML editor of your choice.
2. Find the *UserJourneys* element, and then find the *UserJourney* element with attribute value Id="ProfileEdit".

3. Locate the *OrchestrationStep* element with attribute Order="1" value under this element.
4. Add the following XML snippet under the *ClaimsProviderSelections* node:

```
<ClaimsProviderSelection TargetClaimsExchangeId="TwitterExchange" />
```

5. Locate the *OrchestrationStep* element with attribute Order="2" in the *UserJourney* node and add the following XML snippet under the *ClaimExchanges* node:

```
<ClaimsExchange Id="TwitterExchange" TechnicalProfileReferenceId="Twitter-OAUTH1" />
```

6. Save the XML file.

Uploading the custom policy to your B2C tenant

Proceed with the following steps:

1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
2. Select **Upload Policy**.
3. Check **Overwrite the policy if it exists**.
4. In **Upload policy**, select the *TrustFrameworkExtensions.xml* policy file
5. Click **Upload** and ensure that it does not fail the validation.

Testing the custom policy by using Run Now

Proceed with the following steps:

1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
2. Click the **B2C_1A_signup_signin** custom policy to see its detail view. (You can alternatively select the **B2C_1A_ProfileEdit custom** policy if you also modified above the Profile Edit user journey.) A new blade opens.
3. Click **Run now**.
4. The sign-in page should include the option to sign in with Twitter. You should be able to sign in using your Twitter account.

Pre-filling the requests with a domain hint

Azure AD B2C allows to pre-fill a domain in the requests to the Identity Experience Framework with the **domain_hint** query string parameter. As an illustration, the following request bounces you directly to Facebook for the sign-in.

https://login.microsoftonline.com/litware369b2c.onmicrosoft.com/oauth2/v2.0/authorize?p=B2C_1A_signup_signin&client_id=95b7953b-a499-48b9-bc1b-48bb095d6939&nonce=defaultNonce&redirect_uri=https%3A%2F%2Fjwt.ms&scope=openid&response_type=id_token&domain_hint=facebook.com

Exchanging claims with directories or other systems

This section illustrates how to use in a non-exhaustive manner specific directory provider or REST APIs to exchange claims with your B2C tenant to sustain your own specific scenarios.

The following sections depicts the related configuration for your B2C tenant and related custom policies.

Integrating with your B2C tenant

This section illustrates how to integrate with your B2C tenant to exchange claims.

Adding your B2C tenant as a claim provider

The *B2C_1A_TrustFrameworkBase* custom policy coming with the core templates of the "Starter Pack" already contains the suitable information to declare your B2C tenant as an Azure AD claims provider.

> **Note** For more information, see the second document of this series.

> **Note** For more information on the already declared technical profiles, see the appendix A. of the third document of this series.

So, **the good news is that you don't have anything specific to do to add your B2C as a claims provider.**

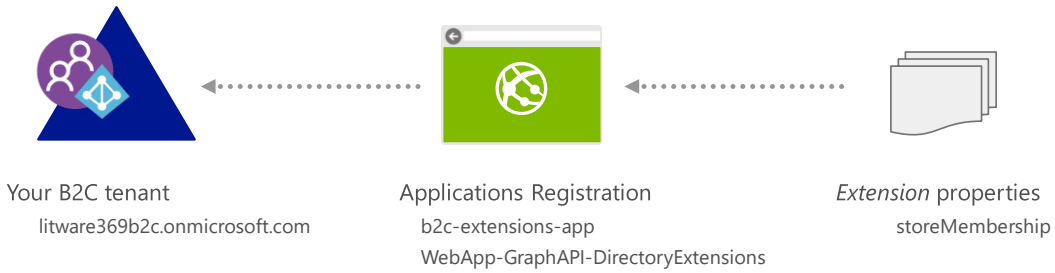
Supporting custom attributes with your B2C tenant

Azure AD B2C allows consumer applications to store some type of custom user profile information. Such a capability is achieved through the support of *Extension* properties (a.k.a. custom attributes) on users.**Note** For more information, see article AZURE ACTIVE DIRECTORY B2C: CREATING AND USING CUSTOM ATTRIBUTES IN A CUSTOM PROFILE EDIT POLICY[34].

If you're not interested in this use case, you can skip this entire section and jump to the next one.

Such *Extension* properties can only be registered on an *Application* object even though they may contain data for a *User*. They belong to that application.

[34] AZURE ACTIVE DIRECTORY B2C: CREATING AND USING CUSTOM ATTRIBUTES IN A CUSTOM PROFILE EDIT POLICY: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-create-custom-attributes-profile-edit-custom



The application must be granted write access to register an *Extension* property. 100 *Extension* properties (across ALL types and ALL applications) can be written to any single object. *Extension* properties are added to the target directory type and becomes immediately accessible in the B2C tenant.

If the application is deleted, those *Extension* properties along with any data contained in them are also removed. If an Extension property is deleted by the application, it is removed on the target directory object, and any data contained in it is removed too.

Thus, before *Extension* properties can be stored, a new application must be created if your B2C tenant has not been upgraded to support the custom policies.

> **Note** For more information, see section § *Managing your users and their attributes with the Graph API* of the third document of this series.

The object id of that application is to be provided in the technical profile(s) of a custom policy.

> **Note** A B2C tenant typically includes a Web App named **b2c-extensions-app**. This application is primarily used by the B2C built-in policies for the custom claims created via the Azure portal and for storing the related *Extension* properties. Using this application to register extensions for custom policies is recommended only for advanced users. You can refer to the aforementioned article for related instructions.

To complete this procedure, proceed with the following steps in order.

1. Create a new application to store the *Extension* properties and get the object id of the considered application.
2. Modify the custom policies to add:
    a. The application's object id.
    b. The new claims type for the *Extension* properties in the custom policies.
    c. The *Extension* properties/custom attributes in the relevant technical profiles of the above policies.
3. Upload the custom policies.
4. Test the custom policies.

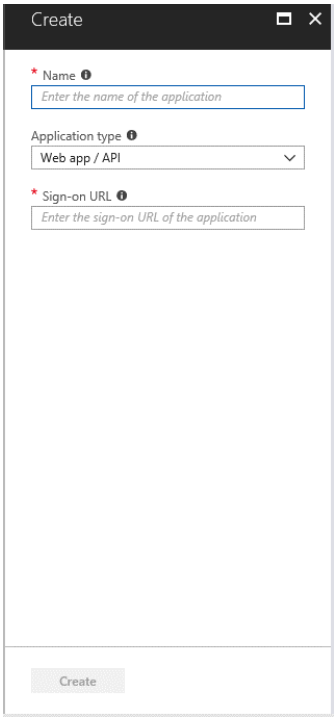
The next sections detail in order each of the above steps.

Creating a new application to store the *Extension* properties

Proceed with the following steps:

1. Log in to the Azure portal as an account with administrative privileges in your Azure tenant.

2. Click **Azure Active Directory** on the left navigation menu. (You may need to find it by selecting **More services>**.)
3. Select **App registrations** and click **New application registration**. A new blade opens.

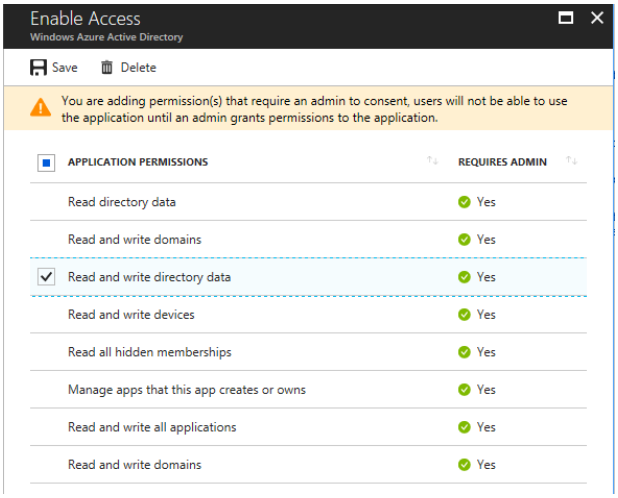


4. Provide the following recommended entries:
    a. In **Name**, enter "*WebApp-GraphAPI-DirectoryExtensions*" for the web application.
    b. For **Application type**, leave **Web app / API** selected.
    c. In **Sign-on URL**, specify "*https://<your_b2c_tenant>.onmicrosoft.com/WebApp-GraphAPI-DirectoryExtensions*" and replace *<your_b2c_tenant>* by the name of your B2C tenant. For example, in our configuration:

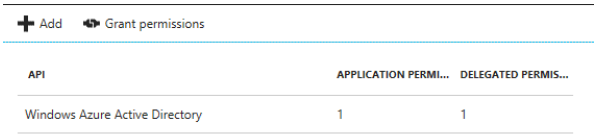
    https://litware369b2c.onmicrosoft.com/WebApp-GraphAPI-DirectoryExtensions
5. Select **Create**.
6. Select **All apps**.

DISPLAY NAME	APPLICATION TYPE	APPLICATION ID
IdentityExperienceFramework	Web app / API	37577079-d70c-4da1-9aff-ce7dd396b444
b2c-extensions-app. Do not modify. Used by AADB2C for storir	Web app / API	1ef57653-176d-4110-b183-7cada64dc888
WebApp-GraphAPI-DirectoryExtensions	Web app / API	4046aed8-66aa-4c9f-9249-7391e8c70244
ProxyIdentityExperienceFramework	Native	b56e4563-c95c-4d8d-aaaf-5692de9a487b

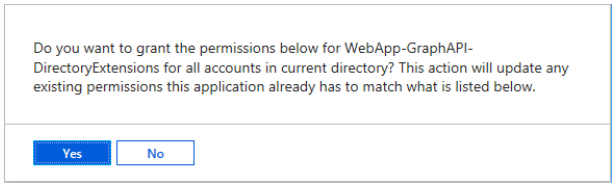
7. Select the newly created web application.
8. Select **Settings** > **Required permissions**.
9. Select **Windows Azure Active Directory** under **API**.



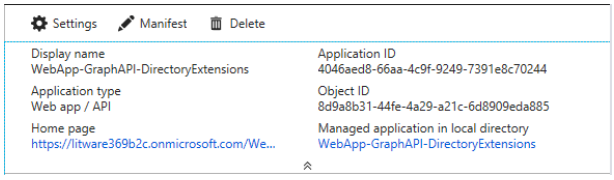
10. Place a checkmark in Application Permissions: **Read and write directory data**, and then select **Save**.



11. Choose **Grant permissions**



12. Confirm **Yes**.
13. Copy to your clipboard and save the following identifiers:



- **Application ID**: 4046aed8-66aa-4c9f-9249-7391e8c70244
- **Object ID**: 8d9a8b31-44fe-4a29-a21c-6d8909eda885

Updating the custom policies

Adding the ApplicationObjectID to the base policy

To add a new claim type, proceed with the following steps:

1. Navigate to the *SocialAndLocalAccounts* folder in the "Starter Pack".

2. Open the extension policy file, i.e. the *TrustFrameworkBase.xml* file, using an XML editor of your choice, for instance Visual Studio (Code).
3. Scroll down to the <TechnicalProfile Id="AAD-Common"> element.
4. Add the following XML snippet after the *Protocol* element.

```
<Metadata>
   <Item Key="ApplicationObjectId">insert objectId here</Item>
   <Item Key="ClientId">insert appId here</Item>
</Metadata>
```

5. In the *Metadata* element underneath, replace "insert objectId here" outlined in red with the above Object ID value, and "insert appId here" outlined in red with the above Application ID value:

```
<ClaimsProviders>
   <ClaimsProvider>
      <DisplayName>Azure Active Directory</DisplayName>
      <TechnicalProfile Id="AAD-Common">
         <DisplayName>Azure Active Directory</DisplayName>
         <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.AzureActiveDirectoryProvider, Web.TPEngine,
                                              Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />

         <Metadata>
           <Item Key="ApplicationObjectId">insert objectId here</Item>
           <Item Key="ClientId">insert appId here</Item>
         </Metadata>

         <CryptographicKeys>
            <Key Id="issuer_secret" StorageReferenceId="TokenSigningKeyContainer" />
         </CryptographicKeys>
         <IncludeInSso>false</IncludeInSso>
         <UseTechnicalProfileForSessionManagement ReferenceId="SM-Noop" />
      </TechnicalProfile>
   </ClaimsProvider>
</ClaimsProviders>
```

6. Save the XML file.

Important note When the above technical profile writes for the first time to the newly created extension property, you may experience a one-time error. The extension property is created the first time it is used.

Adding the new claim type in the custom policies

To add a new claim type, proceed with the following steps:

1. Navigate to the *SocialAndLocalAccounts* folder in the "Starter Pack".
2. Open the extension policy file, i.e. the *TrustFrameworkExtensions.xml* file, using an XML editor of your choice, for instance Visual Studio (Code).
3. In the *ClaimsSchema* node, add the following *ClaimsType* element before the closing </ClaimsSchema> element.

```
<ClaimType Id="extension_storeMembershipNumber">
   <DisplayName>Store Membership Number</DisplayName>
   <DataType>string</DataType>
   <UserHelpText>Your store membership number</UserHelpText>
   <UserInputType>TextBox</UserInputType>
</ClaimType>
```

4. Save the XML file.
5. Add the same *ClaimType* definition to the *TrustFrameworkBase.xml* policy file.

Note Adding a *ClaimType* definition in both the base and the extensions policy files is normally not necessary, However, since the next steps will add the **extension_storeMembershipNumber** to the *TechnicalProfiles* node in the base policy file, the policy validator will reject the upload of the base file without it.

6. Open the *TrustFrameworkBase.xml* file.
7. Add the **extension_storeMembershipNumber** claim:
    a. As an output claim in the technical profile ""to get the value of the claim when a user signs up with a local account.

```
<TechnicalProfile Id="LocalAccountSignUpWithLogonEmail">
   <DisplayName>Email signup</DisplayName>
   <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.SelfAssertedAttributeProvider, Web.TPEngine,
                                        Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
   <Metadata>
      <Item Key="IpAddressClaimReferenceId">IpAddress</Item>
      <Item Key="ContentDefinitionReferenceId">api.localaccountsignup</Item>
      <Item Key="language.button_continue">Create</Item>
   </Metadata>
   <CryptographicKeys>
      <Key Id="issuer_secret" StorageReferenceId="B2C_1A_TokenSigningKeyContainer" />
   </CryptographicKeys>
   <InputClaims>
      <InputClaim ClaimTypeReferenceId="email" />
   </InputClaims>
   <OutputClaims>
      <OutputClaim ClaimTypeReferenceId="objectId" />
      <OutputClaim ClaimTypeReferenceId="email" PartnerClaimType="Verified.Email" Required="true" />
      <OutputClaim ClaimTypeReferenceId="newPassword" Required="true" />
      <OutputClaim ClaimTypeReferenceId="reenterPassword" Required="true" />
      <OutputClaim ClaimTypeReferenceId="executed-SelfAsserted-Input" DefaultValue="true" />
      <OutputClaim ClaimTypeReferenceId="authenticationSource" />
      <OutputClaim ClaimTypeReferenceId="newUser" />

      <!-- Optional claims, to be collected from the user -->
      <OutputClaim ClaimTypeReferenceId="displayName" />
      <OutputClaim ClaimTypeReferenceId="givenName" />
      <OutputClaim ClaimTypeReferenceId="surName" />

      <OutputClaim ClaimTypeReferenceId="extension_storeMembershipNumber"/>

   </OutputClaims>
   <ValidationTechnicalProfiles>
      <ValidationTechnicalProfile ReferenceId="AAD-UserWriteUsingLogonEmail" />
   </ValidationTechnicalProfiles>
   <UseTechnicalProfileForSessionManagement ReferenceId="SM-AAD" />
</TechnicalProfile>
```

    b. As an input and output claim in the technical profile "SelfAsserted-ProfileUpdate" to both retrieve and update the *Extension* property in the user profile for the current user in the directory.

```
<TechnicalProfile Id="SelfAsserted-ProfileUpdate">
   <DisplayName>User ID signup</DisplayName>
   <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.SelfAssertedAttributeProvider, Web.TPEngine,
                                        Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
   <Metadata>
      <Item Key="ContentDefinitionReferenceId">api.selfasserted.profileupdate</Item>
   </Metadata>
   <IncludeInSso>false</IncludeInSso>
   <InputClaims>

      <InputClaim ClaimTypeReferenceId="alternativeSecurityId" />
```

```
      <InputClaim ClaimTypeReferenceId="userPrincipalName" />

      <!-- Optional claims. These claims are collected from the user and can be modified. Any claim added here should be
           updated in the ValidationTechnicalProfile referenced below so it can be written to directory after being
           updated by the user, i.e. AAD-UserWriteProfileUsingObjectId. -->
      <InputClaim ClaimTypeReferenceId="givenName" />
      <InputClaim ClaimTypeReferenceId="surname" />

      <InputClaim ClaimTypeReferenceId="extension_storeMembershipNumber"/>

   </InputClaims>
   <OutputClaims>
      <!-- Required claims -->
      <OutputClaim ClaimTypeReferenceId="executed-SelfAsserted-Input" DefaultValue="true" />

      <!-- Optional claims. These claims are collected from the user and can be modified. Any claim added here should be
           updated in the ValidationTechnicalProfile referenced below so it can be written to directory after being
           updated by the user, i.e. AAD-UserWriteProfileUsingObjectId. -->
      <OutputClaim ClaimTypeReferenceId="givenName" />
      <OutputClaim ClaimTypeReferenceId="surname" />

      <OutputClaim ClaimTypeReferenceId="extension_storeMembershipNumber"/>

   </OutputClaims>
   <ValidationTechnicalProfiles>
      <ValidationTechnicalProfile ReferenceId="AAD-UserWriteProfileUsingObjectId" />
   </ValidationTechnicalProfiles>
</TechnicalProfile>
```

c. As a persisted claim in the technical profile "AAD-UserWriteProfileUsingObjectId" to persist the value of the claim in the *Extension* property, for the current user in the directory.

```
<TechnicalProfile Id="AAD-UserWriteProfileUsingObjectId">
   <Metadata>
     <Item Key="Operation">Write</Item>
     <Item Key="RaiseErrorIfClaimsPrincipalAlreadyExists">false</Item>
     <Item Key="RaiseErrorIfClaimsPrincipalDoesNotExist">true</Item>
   </Metadata>
   <IncludeInSso>false</IncludeInSso>
   <InputClaims>
      <InputClaim ClaimTypeReferenceId="objectId" Required="true" />
   </InputClaims>
   <PersistedClaims>
      <!-- Required claims -->
      <PersistedClaim ClaimTypeReferenceId="objectId" />

      <!-- Optional claims -->
      <PersistedClaim ClaimTypeReferenceId="givenName" />
      <PersistedClaim ClaimTypeReferenceId="surname" />

      <PersistedClaim ClaimTypeReferenceId="extension_storeMembershipNumber" />

   </PersistedClaims>
   <IncludeTechnicalProfile ReferenceId="AAD-Common" />
</TechnicalProfile>
```

d. As an output claim in the technical profile "AAD-UserReadUsingObjectId" to read the value of the *Extension* property every time a user logs in.

```
<TechnicalProfile Id="AAD-UserReadUsingObjectId">
   <Metadata>
      <Item Key="Operation">Read</Item>
      <Item Key="RaiseErrorIfClaimsPrincipalDoesNotExist">true</Item>
   </Metadata>
   <IncludeInSso>false</IncludeInSso>
   <InputClaims>
      <InputClaim ClaimTypeReferenceId="objectId" Required="true" />
   </InputClaims>
   <OutputClaims>
```

```
      <!-- Optional claims -->
      <OutputClaim ClaimTypeReferenceId="signInNames.emailAddress" />
      <OutputClaim ClaimTypeReferenceId="displayName" />
      <OutputClaim ClaimTypeReferenceId="otherMails" />
      <OutputClaim ClaimTypeReferenceId="givenName" />
      <OutputClaim ClaimTypeReferenceId="surname" />

      <OutputClaim ClaimTypeReferenceId="extension_storeMembershipNumber" />

   </OutputClaims>
   <IncludeTechnicalProfile ReferenceId="AAD-Common" />
</TechnicalProfile>
```

At this stage, **only the technical profiles involved in the flow of local accounts only have been changed.**

If the new attribute is desired in the flow of a social/federated account, a different set of technical profiles needs to be changed as illustrated in the three next steps.

e. As an output claim in the technical profile "SelfAsserted-Social" to get the value of the claim when a social/federated user signs up/logs in.

```
<TechnicalProfile Id="SelfAsserted-Social">
   <DisplayName>User ID signup</DisplayName>
   <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.SelfAssertedAttributeProvider, Web.TPEngine,
                                        Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
   <Metadata>
      <Item Key="ContentDefinitionReferenceId">api.selfasserted</Item>
   </Metadata>
   <CryptographicKeys>
      <Key Id="issuer_secret" StorageReferenceId="B2C_1A_TokenSigningKeyContainer" />
   </CryptographicKeys>
   <InputClaims>
      <!-- These claims ensure that any values retrieved in the previous steps (e.g. from an external IDP) are prefilled.
           Note that some of these claims may not have any value, for example, if the external IDP did not provide any of
           these values, or if the claim did not appear in the OutputClaims section of the IDP.
           In addition, if a claim is not in the InputClaims section, but it is in the OutputClaims section, then its
           value will not be prefilled, but the user will still be prompted for it (with an empty value). -->
       <InputClaim ClaimTypeReferenceId="displayName" />
       <InputClaim ClaimTypeReferenceId="givenName" />
       <InputClaim ClaimTypeReferenceId="surname" />
   </InputClaims>
    <OutputClaims>
       <!-- These claims are not shown to the user because their value is obtained through the
            "ValidationTechnicalProfiles" referenced below, or a default value is assigned to the claim. A claim is only
            shown to the user to provide a value if its value cannot be obtained through any other means. -->
       <OutputClaim ClaimTypeReferenceId="objectId" />
       <OutputClaim ClaimTypeReferenceId="newUser" />
       <OutputClaim ClaimTypeReferenceId="executed-SelfAsserted-Input" DefaultValue="true" />

       <!-- Optional claims. These claims are collected from the user and can be modified. If a claim is to be persisted
            in the directory after having been collected from the user, it needs to be added as a PersistedClaim in the
            ValidationTechnicalProfile referenced below, i.e. in AAD-UserWriteUsingAlternativeSecurityId. -->
       <OutputClaim ClaimTypeReferenceId="displayName" />
       <OutputClaim ClaimTypeReferenceId="givenName" />
       <OutputClaim ClaimTypeReferenceId="surname" />

       <OutputClaim ClaimTypeReferenceId="extension_storeMembershipNumber"/>

    </OutputClaims>
    <ValidationTechnicalProfiles>
       <ValidationTechnicalProfile ReferenceId="AAD-UserWriteUsingAlternativeSecurityId" />
    </ValidationTechnicalProfiles>
    <UseTechnicalProfileForSessionManagement ReferenceId="SM-SocialSignup" />
</TechnicalProfile>
```

f. As a persisted claim in the technical profile "AAD-UserWriteUsingAlternativeSecurityId" to persist the value of the claim in the *Extension* property. This is the equivalent of step c for social/federated account.

```
<TechnicalProfile Id="AAD-UserWriteUsingAlternativeSecurityId">
   <Metadata>
     <Item Key="Operation">Write</Item>
     <Item Key="RaiseErrorIfClaimsPrincipalAlreadyExists">true</Item>
     <Item Key="UserMessageIfClaimsPrincipalAlreadyExists">
       You are already registered, please press the back button and sign in instead.
     </Item>
   </Metadata>
   <IncludeInSso>false</IncludeInSso>
   <InputClaimsTransformations>
     <InputClaimsTransformation ReferenceId="CreateOtherMailsFromEmail" />
   </InputClaimsTransformations>
   <InputClaims>
     <InputClaim ClaimTypeReferenceId="AlternativeSecurityId" PartnerClaimType="alternativeSecurityId" Required="true" />
   </InputClaims>
   <PersistedClaims>
      <!-- Required claims -->
      <PersistedClaim ClaimTypeReferenceId="alternativeSecurityId" />
      <PersistedClaim ClaimTypeReferenceId="userPrincipalName" />
      <PersistedClaim ClaimTypeReferenceId="mailNickName" DefaultValue="unknown" />
      <PersistedClaim ClaimTypeReferenceId="displayName" DefaultValue="unknown" />

      <!-- Optional claims -->
      <PersistedClaim ClaimTypeReferenceId="otherMails" />
      <PersistedClaim ClaimTypeReferenceId="givenName" />
      <PersistedClaim ClaimTypeReferenceId="surname" />

      <PersistedClaim ClaimTypeReferenceId="extension_storeMembershipNumber" />

   </PersistedClaims>
   <OutputClaims>
     <OutputClaim ClaimTypeReferenceId="objectId" />
     <OutputClaim ClaimTypeReferenceId="newUser" PartnerClaimType="newClaimsPrincipalCreated" />
     <!-- The following other mails claim is needed for the case when a user is created, we get otherMails from
          directory. Self-asserted provider also has an OutputClaims, and if this is absent, Self-Asserted provider will
          prompt the user for otherMails. -->
     <OutputClaim ClaimTypeReferenceId="otherMails" />
   </OutputClaims>
   <IncludeTechnicalProfile ReferenceId="AAD-Common" />
   <UseTechnicalProfileForSessionManagement ReferenceId="SM-AAD" />
</TechnicalProfile>
```

g. As an output claim in the technical profile "AAD-UserReadUsingAlternativeSecurityId" to persist the value of the claim in the *Extension* property. This is the equivalent of the above step d.

```
<TechnicalProfile Id="AAD-UserReadUsingAlternativeSecurityId">
   <Metadata>
     <Item Key="Operation">Read</Item>
     <Item Key="RaiseErrorIfClaimsPrincipalDoesNotExist">true</Item>
     <Item Key="UserMessageIfClaimsPrincipalDoesNotExist">
       User does not exist. Please sign up before you can sign in.
     </Item>
   </Metadata>
   <InputClaims>
     <InputClaim ClaimTypeReferenceId="AlternativeSecurityId" PartnerClaimType="alternativeSecurityId" Required="true" />
   </InputClaims>
   <OutputClaims>
     <!-- Required claims -->

     <OutputClaim ClaimTypeReferenceId="objectId" />

     <!-- Optional claims -->
     <OutputClaim ClaimTypeReferenceId="userPrincipalName" />
     <OutputClaim ClaimTypeReferenceId="displayName" />
     <OutputClaim ClaimTypeReferenceId="otherMails" />
     <OutputClaim ClaimTypeReferenceId="givenName" />
```

```
      <OutputClaim ClaimTypeReferenceId="surname" />
      <OutputClaim ClaimTypeReferenceId="extension_storeMembershipNumber" />
    </OutputClaims>
    <IncludeTechnicalProfile ReferenceId="AAD-Common" />
</TechnicalProfile>
```

8. Save the XML file.

Declaring the new claim type to the Sign-Up or Sign-In (SUSI) user journey

To declare the new claim type to the Sign-Up or Sign-In (SUSI) user journey, proceed with the following steps:

1. Navigate to the *SocialAndLocalAccounts* folder in the "Starter Pack".
2. Open the extensions policy file, i.e. the *SignUpOrSignIn.xml* file, using an XML editor of your choice, for instance Visual Studio (Code).
3. Find the element <TechnicalProfile Id="PolicyProfile">, and in the *OutputClaims* node add the following XML snippet:

```
<OutputClaim ClaimTypeReferenceId="extension_storeMembershipNumber" />
```

4. Save the XML file.

Registering the new claim type to the Profile Edit user journey

To register the new claim type to the Profile Edit user journey, proceed with the following steps:

1. Still from the *SocialAndLocalAccounts* folder in the "Starter Pack", open the *ProfileEdit.xml* file using an XML editor of your choice.
2. Find the element <TechnicalProfile Id="PolicyProfile">, and in the *OutputClaims* node add the following XML snippet:

```
<OutputClaim ClaimTypeReferenceId="extension_storeMembershipNumber" />
```

3. Save the XML file.

Uploading the custom policies to your B2C tenant

Proceed with the following steps:

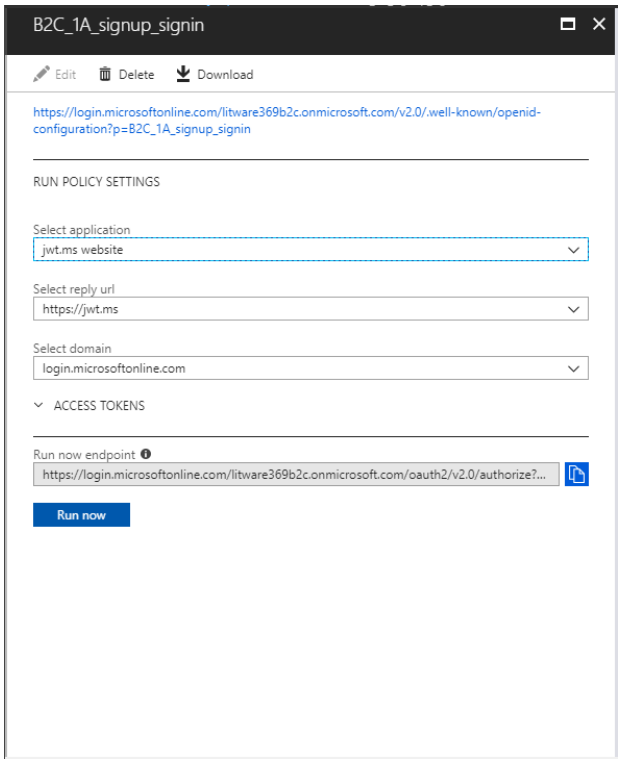
1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
2. Select **Upload Policy**.
3. Check **Overwrite the policy if it exists**.
4. In **Upload policy**, select the *TrustFrameworkBase.xml* policy file.
5. Click **Upload** and ensure that it does not fail the validation.

6. Repeat above steps with the *TrustFrameworkExtensions.xml*, *SignUpOrSignIn.xml*, and the *ProfileEdit.xml* policy files.

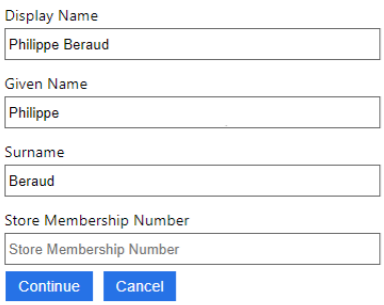
Testing the custom policy by using Run Now

Proceed with the following steps:

1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
2. Click the **B2C_1A_signup_signin** custom policy to see its detail view. A new blade opens.



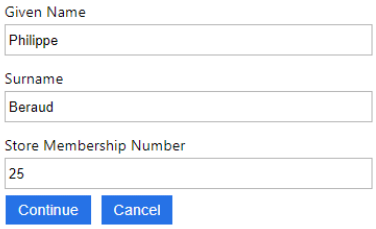
3. Click **Run now**.
4. On the sign in page, select for instance Facebook, and enter your Facebook account's credential when prompted.
5. On the sign-up page, verify that the prompt **Store Membership Number** appears.



6. Specify a value for store membership number and click **Continue**.

7. In the https://jwt.ms page, you should see a claim with type **extension_storeMembershipNumber** that has the value that you specified.

8. Repeat the above steps 2 and 3 with the **B2C_1A_ProfileEdit** custom policy.



9. Change the value of the store membership number and click **Continue**.
10. In the https://jwt.ms page, you should see the new value listed.

Getting the list of all the *Extensions* properties

To get the list of all the *Extensions* properties, a.k.a. custom attributes, that have been created on the **WebApp-GraphAPI-DirectoryExtensions** application, you can use the B2C.exe tool covered in the third document of this series.

Type the following command:

```
PS> .\B2C.exe Get-Extension-Attribute <object-id_of_b2c-extensions-app>
```

In our illustration, 8d9a8b31-44fe-4a29-a21c-6d8909eda885:

```
PS> .\B2C.exe Get-Extension-Attribute 8d9a8b31-44fe-4a29-a21c-6d8909eda885
GET https://graph.windows.net/litware369b2c.onmicrosoft.com/applications/8d9a8b31-44fe-4a29-a21c-6d8909eda885/extensionP
roperties?api-version=1.6
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6IlliUkFRUlljRV9tb3RXVkpLSHJ3TEJiZF85...

200: OK

{
  "odata.metadata": "https://graph.windows.net/litware369b2c.onmicrosoft.com/$metadata#directoryObjects/Microsoft.Direct
oryServices.ExtensionProperty",
  "value": [
    {
      "odata.type": "Microsoft.DirectoryServices.ExtensionProperty",
      "objectType": "ExtensionProperty",
      "objectId": "93ad5eff-76e1-462f-8dde-3cd0ea43faaf",
      "deletionTimestamp": null,
      "appDisplayName": "",
      "name": "extension_4046aed8-66aa-4c9f-9249-7391e8c70244_storeMemebershipNumber",
      "dataType": "String",
      "isSyncedFromOnPremises": false,
      "targetObjects": [
        "User"
      ]
    },
```

The above JSON snippet lists the Extensions property/custom attribute *storeMembershipNumber* created in the previous sections: *extension*_4046aed8-66aa-4c9f-9249-7391e8c70244_*storeMembershipNumber*

As you can see, the custom attributes are stored in the format:

extension_<Client ID>_<attribute>

Where:

- <*Client ID*> is the "*Client ID*" value of the **WebApp-GraphAPI-DirectoryExtensions** application: 4046aed8-66aa-4c9f-9249-7391e8c70244.
- <*attribute*> is the attribute name.

You can then update one of the two above *.json* files with the new property and a value for the property and run the following:

```
PS> .\B2C Update-User <object_id_of_user> <path_to_json_file>
```

Sharing the same Extension properties with the built-in custom policies

As noticed earlier, when you add *Extension* properties (a.k.a. custom attributes) via the Azure portal for built-in policies, those properties are registered using the **b2c-extensions-app** that exists in every B2C tenant.

To use these *Extension* properties in your custom policies, proceed with the following steps:

1. Within your B2C tenant in the Azure portal, navigate to **Azure Active Directory** and select **App registrations**.

2. Find your **b2c-extensions-app** and select it.
3. Under **Essentials**, record - as before with the **WebApp-GraphAPI-DirectoryExtensions** app - both the **Application ID** and the **Object ID**.
4. Open the base policy file, specify them as before in the technical profile "AAD-Common", save the XML file and upload it to your B2C tenant.

Integrating with a RESTful API

The Identity Experience Framework in Azure AD B2C provides complete control for configuring policies. It enables you to integrate identity providers with external REST full APIs through user journeys. IEF sends and receives data in form of claims. You can configure a custom policy to exchange claims and perform complex validation of identity information through a REST API.

If you're not interested in this use case, you can skip this entire section and jump to the next one.

You can design the integration with a RESTful API in the following two ways:

1. **Validation technical profile**: The call to the RESTful API happens within the validation technical profile of the specified technical profile. The validation technical profile validates the user-provided data before the user journey moves forward. With the validation technical profile, you can:
    a. Send input claims.
    b. Validate the input claims and throw custom error messages.
    c. Send back output claims.
2. **Claims exchange**: This design is similar to the validation technical profile, but it happens within an orchestration step. This definition is limited to:
    a. Send input claims.
    b. Send back output claims.

We will illustrate both ways.

Creating an API application

Follow the instructions provided in section § *Building a RESTful API claims provider* in the Appendix.

Whilst this section features Azure App Service - Web Apps[35] and the ASP.Net Core technology, one should note that Azure Functions[36] also provides an excellent foundation to create (serverless) RESTful APIs in the cloud.

Integrating with a RESTful API for validating user input

This section shows how to create a user journey that interacts with the above RESTful API. It will leverage this RESTful service that validates user input and sends an error message if the input data is not valid. The data that the service validates is part of the identity information provided when a user signs up to your application.

> **Note** For more information, see articles WALKTHROUGH: INTEGRATE REST API CLAIMS EXCHANGES IN YOUR AZURE AD B2C USER JOURNEY AS VALIDATION ON USER INPUT[37] and INTEGRATE REST API CLAIMS EXCHANGES IN YOUR AZURE AD B2C USER JOURNEY AS VALIDATION OF USER INPUT[38].

If you're not interested in this use case, you can skip this entire section and jump to the next one.

To do so, you will have to:

1. Update custom policies to use the API as claim provider in the intended user journey(s).
2. Upload custom policies.
3. Test the custom policy using **Run Now**.

The next sections detail the above.

Updating the custom policies

Proceed with the following steps:

1. Navigate to the *SocialAndLocalAccounts* folder in the "Starter Pack".
2. Open the base policy file, i.e. the *TrustFrameworkBase.xml* file, using an XML editor of your choice, for instance Visual Studio (Code).
3. Replace the <> element with the following XML snippet in the technical profiles "LocalAccountSignUpWithLogonEmail" and "SelfAsserted-Social".

```
<OutputClaim ClaimTypeReferenceId="extension_storeMembershipNumber" PartnerClaimType="storeMembershipNumber" />
```

4. Save the XML file.
5. Now open the extension policy file, i.e. the *TrustFrameworkExtensions.xml* file.

---

[35] Azure App Service - Web Apps: https://azure.microsoft.com/en-us/services/app-service/web/
[36] Azure Functions: https://azure.microsoft.com/en-us/services/functions/
[37] WALKTHROUGH: INTEGRATE REST API CLAIMS EXCHANGES IN YOUR AZURE AD B2C USER JOURNEY AS VALIDATION ON USER INPUT: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-rest-api-validation-custom
[38] INTEGRATE REST API CLAIMS EXCHANGES IN YOUR AZURE AD B2C USER JOURNEY AS VALIDATION OF USER INPUT: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-custom-rest-api-netfw

6. In the *ClaimsProviders* node, add the following *ClaimsProvider* element.

```
<ClaimsProvider>
   <DisplayName>REST APIs</DisplayName>
   <TechnicalProfiles>
      <TechnicalProfile Id="ValidateStoreMembershipNumber">
         <DisplayName>Validate Store Membership Number</DisplayName>
         <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.RestfulProvider, Web.TPEngine, Version=1.0.0.0,
                                               Culture=neutral, PublicKeyToken=null" />
         <Metadata>
            <Item Key="ServiceUrl">https://your_webservice.azurewebsites.net/api/membership/validate</Item>
            <Item Key="AuthenticationType">None</Item>
            <Item Key="SendClaimsIn">Body</Item>
         </Metadata>
         <InputClaims>
            <InputClaim ClaimTypeReferenceId="extension_storeMembershipNumber"
                        PartnerClaimType="storeMembershipNumber" />
         </InputClaims>
         <UseTechnicalProfileForSessionManagement ReferenceId="SM-Noop" />
      </TechnicalProfile>
      <TechnicalProfile Id="LocalAccountSignUpWithLogonEmail">
         <ValidationTechnicalProfiles>
            <ValidationTechnicalProfile ReferenceId="ValidateStoreMembershipNumber" />
         </ValidationTechnicalProfiles>
      </TechnicalProfile>
      <TechnicalProfile Id="SelfAsserted-Social">
         <ValidationTechnicalProfiles>
            <ValidationTechnicalProfile ReferenceId="ValidateStoreMembershipNumber" />
         </ValidationTechnicalProfiles>
      </TechnicalProfile>
      <TechnicalProfile Id="SelfAsserted-ProfileUpdate">
         <ValidationTechnicalProfiles>
            <ValidationTechnicalProfile ReferenceId="ValidateStoreMembershipNumber" />
         </ValidationTechnicalProfiles>
      </TechnicalProfile>
   </TechnicalProfiles>
</ClaimsProvider>
```

The technical profile "ValidateStoreMembershipNumber" is used as a validation technical profile for the technical profiles "LocalAccountSignUpWithLogonEmail", "SelfAsserted-Social", and "SelfAsserted-ProfileUpdate" to validate the user input for the *Extension* property **extension_storeMembershipNumber**.

7. Replace your_webservice highlighted in red with the name under which your RESTful API has been published. For example, in our configuration:

   "storemembershipapi20180627033636", See section § *Publishing the project as an Azure Website* in the Appendix.

8. Save the XML file.

Important note For the sake of simplicity, the above RESTful API is provided with no authentication, see related metadata in the above technical profile ValidateStoreMembershipNumber. For more information about how to secure such an API, see articles SECURE YOUR RESTFUL SERVICES BY USING HTTP BASIC AUTHENTICATION[39] and SECURE YOUR RESTFUL SERVICE BY USING CLIENT CERTIFICATES[40] to support a basic authentication respectively a certificate-based authentication. The policy files along with the code samples provided in the *folder scenarios\ aadb2c-ief-rest-api-*

[39] SECURE YOUR RESTFUL SERVICES BY USING HTTP BASIC AUTHENTICATION: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-custom-rest-api-netfw-secure-basic

[40] SECURE YOUR RESTFUL SERVICE BY USING CLIENT CERTIFICATES: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-custom-rest-api-netfw-secure-basic

netfw-secure-basic respectively *scenarios\ aadb2c-ief-rest-api-netfw-secure-cert* of the "Starter Pack can be used as an illustration or as a starting point.

Uploading the custom policies to your B2C tenant

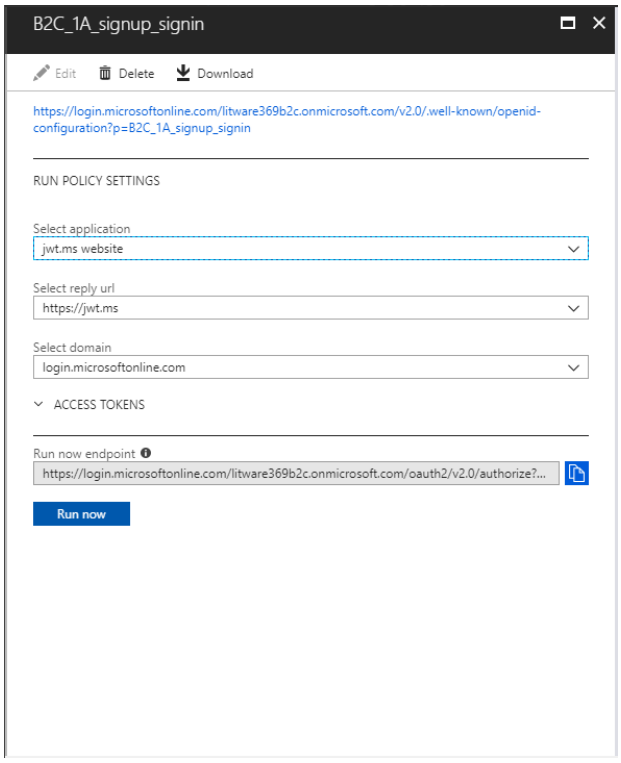
Proceed with the following steps:

1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
2. Select **Upload Policy**.
3. Check **Overwrite the policy if it exists**.
4. In **Upload policy**, select the *TrustFrameworkBase.xml* policy file.
5. Click **Upload** and ensure that it does not fail the validation.
6. Repeat above steps with the *TrustFrameworkExtensions.xml* policy file.

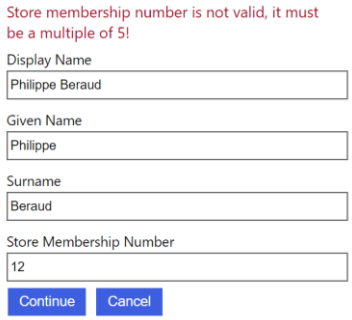
Testing the custom policy by using Run Now

To test the custom policy, proceed with the following steps:

1. In the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
2. Select **Users**.
3. In **All users**, select the user you created in above section § *Integrating with your B2C tenant*, and select **Delete user**. Select **Yes** to confirm the deletion.
4. Navigate to the previous page.
5. Click the **B2C_1A_signup_signin** custom policy to see its detail view. A new blade opens.

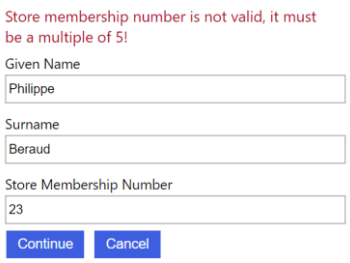


6. Click **Run now**.
7. On the sign in page, select **Sign up now**.
8. On the sign-up page, verify that the prompt **Store Membership Number** appears.
9. Attempt to sign up with a store membership number that is not a multiple of 5.



10. Verify that if you specify a multiple of 5 for the store membership number then sign up is successful. In the https://jwt.ms page, you should see a claim with type **extension_storeMembershipNumber** that has the value that you specified as a multiple of 5.

11. Repeat the above steps 5 and 6 with the **B2C_1A_ProfileEdit** custom policy.
12. Change the value of the store membership number to another value that is not a multiple of 5 and click **Continue**.



13. Now specify a multiple of 5 for the store membership number and click **Continue**. In the https://jwt.ms page, you should see a claim with type **extension_storeMembershipNumber** that has the value that you specified as a multiple of 5.

Integrating with a RESTful API for validating user input

This section shows how to create a user journey that interacts with the previous RESTful API (See section § *Creating an API application*). It will leverage this RESTful service that also reads the input user claims and returns a Store Membership Date.

> **Note** For more information on how integrating a REST API for claims exchange, see article WALKTHROUGH: INTEGRATE REST API CLAIMS EXCHANGES IN YOUR AZURE AD B2C USER JOURNEY AS AN ORCHESTRATION STEP[41].

If you're not interested in this use case, you can skip this entire section and jump to the next one.

To do so, you will have to:

1. Update custom policies to use the API as claim provider in the intended user journey(s).
2. Upload custom policies.
3. Test the custom policy using **Run Now**.

The next sections detail the above.

[41] WALKTHROUGH: INTEGRATE REST API CLAIMS EXCHANGES IN YOUR AZURE AD B2C USER JOURNEY AS AN ORCHESTRATION STEP: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-rest-api-step-custom

Updating the custom policies

Proceed with the following steps:

1. Navigate to the *SocialAndLocalAccounts* folder in the "Starter Pack".
2. Open the extension policy file, i.e. the *TrustFrameworkExtensions.xml* file, using an XML editor of your choice, for instance Visual Studio (Code).
3. In the *BuildingBlocks* node, insert the following *ClaimType* to the collection in the *ClaimsSchema* node.

```
<ClaimType Id="extension_storeMembershipDate">
  <DisplayName>Store Membership Date</DisplayName>
  <DataType>string</DataType>
  <UserHelpText>Your store membership date</UserHelpText>
  <UserInputType>TextBox</UserInputType>
</ClaimType>
```

4. In the *ClaimsProviders* node, find the *ClaimsProvider* node that contains the <DisplayName>REST APIs</DisplayName> element.
5. Add the following *TechnicalProfile* element to this claims provider.

```
<!-- Obtain claim technical profile -->
<TechnicalProfile Id="ObtainStoreMembershipDate">
   <DisplayName>Obtain Store Membership Date</DisplayName>
   <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.RestfulProvider, Web.TPEngine, Version=1.0.0.0,
Culture=neutral,
                                        PublicKeyToken=null" />
   <Metadata>
      <Item Key="ServiceUrl">https://your_webservice.azurewebsites.net/api/membership/membershipdate</Item>
      <Item Key="AuthenticationType">None</Item>
      <Item Key="SendClaimsIn">Body</Item>
   </Metadata>
   <InputClaims>
      <InputClaim ClaimTypeReferenceId="extension_storeMembershipNumber" PartnerClaimType="storeMembershipNumber" />
   </InputClaims>
   <OutputClaims>
      <OutputClaim ClaimTypeReferenceId="extension_storeMembershipDate" PartnerClaimType="storeMembershipDate" />
   </OutputClaims>
   <UseTechnicalProfileForSessionManagement ReferenceId="SM-Noop" />
</TechnicalProfile>
```

6. As before, replace your_webservice highlighted in red with the name under which your RESTful API has been published. For example, in our configuration:

   "storemembershipapi20180627033636"
   See section § *Publishing the project as an Azure Website* in the Appendix.

7. Scroll down to the *UserJourneys* node at the end of the file and add the following *UserJourney* element to this node.

```
<UserJourney Id="ProfileEditObtainApiClaim">
   <OrchestrationSteps>
      <OrchestrationStep Order="1" Type="ClaimsProviderSelection" ContentDefinitionReferenceId="api.idpselections">
         <ClaimsProviderSelections>
            <ClaimsProviderSelection TargetClaimsExchangeId="LocalAccountSigninEmailExchange" />
         </ClaimsProviderSelections>
      </OrchestrationStep>
      <OrchestrationStep Order="2" Type="ClaimsExchange">
         <ClaimsExchanges>
```

```
            <ClaimsExchange Id="LocalAccountSigninEmailExchange"
                            TechnicalProfileReferenceId="SelfAsserted-LocalAccountSignin-Email" />
         </ClaimsExchanges>
      </OrchestrationStep>
      <OrchestrationStep Order="3" Type="ClaimsExchange">
         <ClaimsExchanges>
            <ClaimsExchange Id="AADUserReadWithObjectId" TechnicalProfileReferenceId="AAD-UserReadUsingObjectId" />
         </ClaimsExchanges>
      </OrchestrationStep>
      <OrchestrationStep Order="4" Type="ClaimsExchange">
         <ClaimsExchanges>
            <ClaimsExchange Id="B2CUserProfileUpdateExchange" TechnicalProfileReferenceId="SelfAsserted-ProfileUpdate" />
         </ClaimsExchanges>
      </OrchestrationStep>
      <OrchestrationStep Order="5" Type="ClaimsExchange">
         <ClaimsExchanges>
           <ClaimsExchange Id="GetStoreMembershipDateData" TechnicalProfileReferenceId="ObtainStoreMembershipDate" />
         </ClaimsExchanges>
      </OrchestrationStep>
      <OrchestrationStep Order="6" Type="SendClaims" CpimIssuerTechnicalProfileReferenceId="JwtIssuer" />
   </OrchestrationSteps>
   <ClientDefinition ReferenceId="DefaultWeb" />
</UserJourney>
```

8. Save the XML file.
9. Now open the *ProfileEdit.xml* file, using an XML editor of your choice, for instance Visual Studio (Code).
10. Scroll down to the *RelyingParty* node and under this node replace the *DefaultUserJourney* node with the following XML element.

```
<DefaultUserJourney ReferenceId="ProfileEditObtainApiClaim" />
```

11. Under the <TechnicalProfile Id="PolicyProfile"> node, add the following XML snippet below the *OutputClaims* node.

```
<OutputClaim ClaimTypeReferenceId="extension_storeMembershipDate" />
```

12. Save the XML file.

Uploading the custom policies to your B2C tenant

Proceed with the following steps:

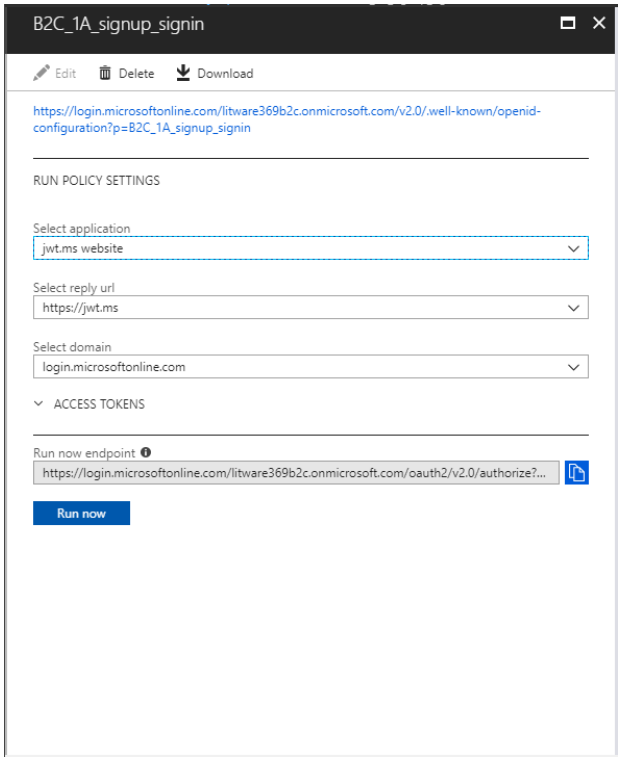
1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
2. Select **Upload Policy**.
3. Check **Overwrite the policy if it exists**.
4. In **Upload policy**, select the *TrustFrameworkExtensions.xml* policy file.
5. Click **Upload** and ensure that it does not fail the validation.
6. Repeat the above steps with the *ProfileEdit.xml* policy file.

Testing the custom policy

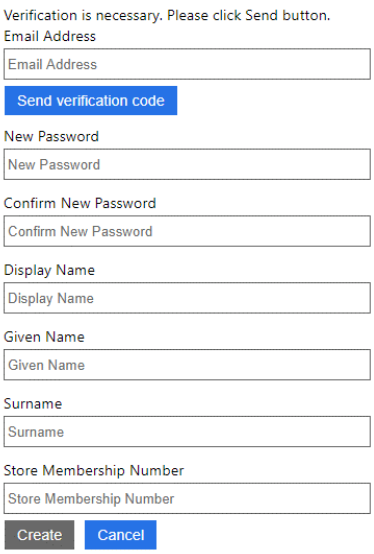
To test the custom policy, proceed with the following steps:

1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.

2. Use the **Users** blade to delete the user you created in the previous test.
3. Close the **Users** blade and then click the **B2C_1A_signup_signin** custom policy to see its detail view. A new blade opens.



4. Click **Run now**.
5. On the sign in page, select **Sign up now**.



6. On the sign-up page, sign up again.
7. When sign up is complete, click the **B2C_1A_EditProfile** custom policy to edit your profile and click **Run now**.

8. Log in with the previously created account. After updating your profile, you should see the **storeMembershipDate** claim in the token sent back from the Identity Experience Framework in Azure AD B2C.

Implementing a custom user journey

This section covers how to:

- Create a new user journey.
- Customize an existing user journey.
- Customize the UI of a user journey.

The next sections detail the above in order.

Creating a new user journey

This section illustrates how to modify an existing user journey to create a new one.

If you're not interested in this use case, you can skip this entire section and jump to the next one.

As such, and for the sake of the illustration, we will use the SignUpOrSignin user journey that comes with the "Starter Pack" in the *SocialAndLocalAccountsWithMfa* folder in its *base* policy file to create as such another user journey to sign in a user without multi-factor authentication (MFA); so, namely the eponym one in the *SocialAndLocalAccounts* folder. This user journey will be named **SignUpOrSignin_NoMfa**.

Proceed with the follow steps:

1. Navigate to the folder in the "Starter Pack".
2. Open the base policy file, i.e. the *TrustFrameworkBase.xml* file, using an XML editor of your choice, for instance Visual Studio (Code).
3. Scroll down and locate the SignUpOrSignin user journey.
4. Copy the XML between <UserJourney> and </UserJourney> XML elements and paste it as an independent user journey in the same file.
5. Rename the *Id* attribute on the copied *UserJourney* XML element to **SignUpOrSignin_NoMfa** or something similar.

```
<UserJourney Id="SignUpOrSignin_NoMfa">
  <OrchestrationSteps>

    <OrchestrationStep Order="1" Type="CombinedSignInAndSignUp" ContentDefinitionReferenceId="api.signuporsignin">
      <ClaimsProviderSelections>
        <ClaimsProviderSelection TargetClaimsExchangeId="FacebookExchange" />
        <ClaimsProviderSelection ValidationClaimsExchangeId="LocalAccountSigninEmailExchange" />
      </ClaimsProviderSelections>
      <ClaimsExchanges>
        <ClaimsExchange Id="LocalAccountSigninEmailExchange"
                        TechnicalProfileReferenceId="SelfAsserted-LocalAccountSignin-Email" />
      </ClaimsExchanges>
    </OrchestrationStep>

    <!-- Check if the user has selected to sign in using one of the social providers -->
    <OrchestrationStep Order="2" Type="ClaimsExchange">
      <Preconditions>
        <Precondition Type="ClaimsExist" ExecuteActionsIf="true">
          <Value>objectId</Value>
          <Action>SkipThisOrchestrationStep</Action>
        </Precondition>
```

```
      </Preconditions>
      <ClaimsExchanges>
        <ClaimsExchange Id="FacebookExchange" TechnicalProfileReferenceId="Facebook-OAUTH" />
        <ClaimsExchange Id="SignUpWithLogonEmailExchange"
                        TechnicalProfileReferenceId="LocalAccountSignUpWithLogonEmail" />
      </ClaimsExchanges>
    </OrchestrationStep>

    <!-- For social IDP authentication, attempt to find the user account in the directory. -->
    <OrchestrationStep Order="3" Type="ClaimsExchange">
      <Preconditions>
        <Precondition Type="ClaimEquals" ExecuteActionsIf="true">
          <Value>authenticationSource</Value>
          <Value>localAccountAuthentication</Value>
          <Action>SkipThisOrchestrationStep</Action>
        </Precondition>
      </Preconditions>
      <ClaimsExchanges>
        <ClaimsExchange Id="AADUserReadUsingAlternativeSecurityId"
                        TechnicalProfileReferenceId="AAD-UserReadUsingAlternativeSecurityId-NoError" />
      </ClaimsExchanges>
    </OrchestrationStep>

    <!-- Show self-asserted page only if the directory does not have the user account already (i.e. we do not have an
         objectId). This can only happen when authentication happened using a social IDP. If local account was created
         or authentication done using ESTS in step 2, then an user account must exist in the directory by this time.
    -->
    <OrchestrationStep Order="4" Type="ClaimsExchange">
      <Preconditions>
        <Precondition Type="ClaimsExist" ExecuteActionsIf="true">
          <Value>objectId</Value>
          <Action>SkipThisOrchestrationStep</Action>
        </Precondition>
      </Preconditions>
      <ClaimsExchanges>
        <ClaimsExchange Id="SelfAsserted-Social" TechnicalProfileReferenceId="SelfAsserted-Social" />
      </ClaimsExchanges>
    </OrchestrationStep>

    <!-- This step reads any user attributes that we may not have received when authenticating using ESTS so they can
         be sent in the token. -->
    <OrchestrationStep Order="5" Type="ClaimsExchange">
      <Preconditions>
        <Precondition Type="ClaimEquals" ExecuteActionsIf="true">
          <Value>authenticationSource</Value>
          <Value>socialIdpAuthentication</Value>
          <Action>SkipThisOrchestrationStep</Action>
         </Precondition>
      </Preconditions>
      <ClaimsExchanges>
        <ClaimsExchange Id="AADUserReadWithObjectId" TechnicalProfileReferenceId="AAD-UserReadUsingObjectId" />
      </ClaimsExchanges>
    </OrchestrationStep>
    <!-- The previous step (SelfAsserted-Social) could have been skipped if there were no attributes to collect
          from the user. So, in that case, create the user in the directory if one does not already exist
          (verified using objectId which would be set from the last step if account was created in the directory. -->
    <OrchestrationStep Order="6" Type="ClaimsExchange">
      <Preconditions>
        <Precondition Type="ClaimsExist" ExecuteActionsIf="true">
          <Value>objectId</Value>
          <Action>SkipThisOrchestrationStep</Action>
        </Precondition>
      </Preconditions>
      <ClaimsExchanges>
        <ClaimsExchange Id="AADUserWrite" TechnicalProfileReferenceId="AAD-UserWriteUsingAlternativeSecurityId" />
      </ClaimsExchanges>
    </OrchestrationStep>

    <!-- Phone verification: If MFA is not required, the next two steps (#7-#8) should be removed.
         This step checks whether there's a phone number on record,  for the user. If found, then the user is challenged
         to verify it. -->
    <OrchestrationStep Order="7" Type="ClaimsExchange">
      <Preconditions>
        <Precondition Type="ClaimsExist" ExecuteActionsIf="true">
          <Value>isActiveMFASession</Value>
          <Action>SkipThisOrchestrationStep</Action>
        </Precondition>
      </Preconditions>
```

```
        <ClaimsExchanges>
          <ClaimsExchange Id="PhoneFactor-Verify" TechnicalProfileReferenceId="PhoneFactor-InputOrVerify" />
        </ClaimsExchanges>
      </OrchestrationStep>

      <!-- Save MFA phone number: The precondition verifies whether the user provided a new number in the
           previous step. If so, then the phone number is stored in the directory for future authentication
           requests. -->
      <OrchestrationStep Order="8" Type="ClaimsExchange">
        <Preconditions>
          <Precondition Type="ClaimsExist" ExecuteActionsIf="false">
            <Value>newPhoneNumberEntered</Value>
            <Action>SkipThisOrchestrationStep</Action>
          </Precondition>
        </Preconditions>
        <ClaimsExchanges>
          <ClaimsExchange Id="AADUserWriteWithObjectId"
                          TechnicalProfileReferenceId="AAD-UserWritePhoneNumberUsingObjectId" />
        </ClaimsExchanges>
      </OrchestrationStep>
```

```
      <OrchestrationStep Order="9" Type="SendClaims" CpimIssuerTechnicalProfileReferenceId="JwtIssuer" />

    </OrchestrationSteps>
    <ClientDefinition ReferenceId="DefaultWeb" />
</UserJourney>
```

6. The *OrchestrationStep* XML elements with Order = 7 to 8 above perform phone authentication. Remove those steps.
7. Change the order of the last orchestration step from 9 to 7. If you don't do that and later attempt to use the policy, Azure AD B2C will complain about inconsistent order numbers.
8. Save the XML file.
9. In the same folder, copy the *SignUpOrSignin.xml* policy to the *SignUpOrSignin_NoMfa.xml* file or something similar and open that XML file.
10. Change the *PolicyId* attribute on *TrustFrameworkPolicy* XML element to **B2C_1A_signup_signin_nomfa**. When a policy is uploaded, the *PolicyId* attribute is used to determine its name in the system.
11. Similarly, change the *ReferenceId* attribute of the *DefaultUserJourney* XML element under the *RelyingParty* section to **SignUpOrSignin_NoMfa**. This indicates which user journey will be executed when a request comes with **p=SignUpOrSignin_NoMfa**.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<TrustFrameworkPolicy
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:xsd="http://www.w3.org/2001/XMLSchema"
          xmlns="http://schemas.microsoft.com/online/cpim/schemas/2013/06"
          PolicySchemaVersion="0.3.0.0"
          TenantId="litware369b2c.onmicrosoft.com"
```

```
          PolicyId="B2C_1A_signup_signin_nomfa"
```

```
          PublicPolicyUri="http://litware369b2c.onmicrosoft.com/B2C_1A_signup_signin">
  <BasePolicy>
    <TenantId>litware369b2c.onmicrosoft.com</TenantId>
    <PolicyId>B2C_1A_TrustFrameworkExtensions</PolicyId>
  </BasePolicy>

  <RelyingParty>
```

```
    <DefaultUserJourney ReferenceId="SignUpOrSignIn_NoMfa" />
```

```
    <TechnicalProfile Id="PolicyProfile">
      <DisplayName>PolicyProfile</DisplayName>
```

```
      <Protocol Name="OpenIdConnect" />
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="displayName" />
        <OutputClaim ClaimTypeReferenceId="givenName" />
        <OutputClaim ClaimTypeReferenceId="surname" />
        <OutputClaim ClaimTypeReferenceId="email" />
        <OutputClaim ClaimTypeReferenceId="objectId" PartnerClaimType="sub"/>
        <OutputClaim ClaimTypeReferenceId="identityProvider" />
      </OutputClaims>
      <SubjectNamingInfo ClaimType="sub" />
    </TechnicalProfile>
  </RelyingParty>
</TrustFrameworkPolicy>
```

12. Save the XML file
13. Upload above XML files to your B2C tenant as before.
14. Select the **B2C_1A_signup_signin_nomfa** newly uploaded policy and click **Run Now**.

This should allow an existing user to sign-up or sign in without phone verification.

Customizing an existing user journey

This section illustrates some common customization for a user journey.

Note For more information, see article AZURE ACTIVE DIRECTORY B2C: MODIFY SIGN UP TO ADD NEW CLAIMS AND CONFIGURE USER INPUT[42].

If you're not interested in this use case, you can skip this entire section and jump to the next one.

Collecting a new attribute from the user during sign up and send it to the application

This section depicts how to use a newly created attribute in a sign-up or sign in policy to collect it from the user, and ultimately send the collected value to the consumer application that runs the policy.

This twofold process consists in:

1. Adding a claim type for the new attribute in the claims schema section,
2. And then adding the attribute in the self-asserted provider to collect from the user in local account creation at sign-up time,

As an illustration in order of the above process, proceed with the following steps:

1. Navigate this time to the folder in the "Starter Pack".
2. Open the base policy file, i.e. the *TrustFrameworkBase.xml* file, using an XML editor of your choice, for instance Visual Studio (Code).
3. Scroll down to the *ClaimsSchema* section to add a claim type for the new attribute.
4. Copy the **givenName** claim type that is close to the end of the *ClaimsSchema* section, and then paste it just before the end of the *ClaimsSchema* section.

[42] AZURE ACTIVE DIRECTORY B2C: MODIFY SIGN UP TO ADD NEW CLAIMS AND CONFIGURE USER INPUT: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-configure-signup-self-asserted-custom

5. In the pasted claim type:

    a. Update the *Id* attribute to a unique value so it can be later referenced from other places in the custom policy. For example, in our illustration, **surname**.
    b. Update the other elements to provide appropriate values. One should note that *DisplayName* and *UserHelpText* XML elements are shown to the end user.
    c. Eventually update *DefaultPartnerClaimTypes* XML element to reflect the *default* claim types that you expect to go in the token to the consumer application.

```
<!-- SECTION III: Additional claims that can be collected from the users, stored in the directory, and sent in the token.
     Add additional claims here. -->

<ClaimType Id="givenName">
…
</ClaimType>
```

```
<ClaimType Id="surname">
   <DisplayName>Surname</DisplayName>
   <DataType>string</DataType>
   <DefaultPartnerClaimTypes>
      <Protocol Name="OAuth2" PartnerClaimType="family_name" />
      <Protocol Name="OpenIdConnect" PartnerClaimType="family_name" />
      <Protocol Name="SAML2" PartnerClaimType="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname" />
   </DefaultPartnerClaimTypes>
   <UserHelpText>Your surname (also known as family name or last name.</UserHelpText>
   <UserInputType>TextBox</UserInputType>
</ClaimType>
```

6. Now that the new claim type is fully specified, locate the **SignUpOrSignIn** user journey in the same policy file.

```
<UserJourney Id="SignUpOrSignIn">
   <OrchestrationSteps>

      <OrchestrationStep Order="1" Type="CombinedSignInAndSignUp" ContentDefinitionReferenceId="api.signuporsignin">
         <ClaimsProviderSelections>
            <ClaimsProviderSelection TargetClaimsExchangeId="FacebookExchange" />
            <ClaimsProviderSelection ValidationClaimsExchangeId="LocalAccountSigninEmailExchange" />
         </ClaimsProviderSelections>
         <ClaimsExchanges>
            <ClaimsExchange Id="LocalAccountSigninEmailExchange"
                            TechnicalProfileReferenceId="SelfAsserted-LocalAccountSignin-Email" />
         </ClaimsExchanges>
      </OrchestrationStep>

      <!-- Check if the user has selected to sign in using one of the social providers -->
      <OrchestrationStep Order="2" Type="ClaimsExchange">
         <Preconditions>
            <Precondition Type="ClaimsExist" ExecuteActionsIf="true">
               <Value>objectId</Value>
               <Action>SkipThisOrchestrationStep</Action>
            </Precondition>
         </Preconditions>
         <ClaimsExchanges>
            <ClaimsExchange Id="FacebookExchange" TechnicalProfileReferenceId="Facebook-OAUTH" />
```

```
            <ClaimsExchange Id="SignUpWithLogonEmailExchange"
                            TechnicalProfileReferenceId="LocalAccountSignUpWithLogonEmail" />
```

```
         </ClaimsExchanges>
      </OrchestrationStep>
    …
  </OrchestrationSteps>
  <ClientDefinition ReferenceId="DefaultWeb" />
</UserJourney>
```

7. In the orchestration step 2, locate the technical profile name for the **SignUpWithLogonEmailExchange** claims exchange. It should be the **LocalAccountSignUpWithLogonEmail** technical profile. If you recall What we've covered in section § *Integrating with your B2C tenant*, the **LocalAccountSignUpWithLogonEmail** self-asserted technical profile was used during local account creation.
8. Search that technical profile in the policy file.
9. In the *OutputClaims* section of the technical profile, add the newly created claim type in steps 4 to 5.

   Essentially an output claim (i.e. an *OutputClaim* XML element) listed in such a technical profile indicates that this claim needs to be sent back by the provider and thus will be sourced from the user.

```
<TechnicalProfile Id="LocalAccountSignUpWithLogonEmail">
   <DisplayName>Email signup</DisplayName>
   <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.SelfAssertedAttributeProvider, Web.TPEngine,
                                        Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
    …
   <OutputClaims>
      <OutputClaim ClaimTypeReferenceId="objectId" />
      <OutputClaim ClaimTypeReferenceId="email" PartnerClaimType="Verified.Email" Required="true" />
      <OutputClaim ClaimTypeReferenceId="newPassword" Required="true" />
      <OutputClaim ClaimTypeReferenceId="reenterPassword" Required="true" />
      <OutputClaim ClaimTypeReferenceId="executed-SelfAsserted-Input" DefaultValue="true" />
      <OutputClaim ClaimTypeReferenceId="authenticationSource" />
      <OutputClaim ClaimTypeReferenceId="newUser" />

      <!-- Optional claims, to be collected from the user -->
      <OutputClaim ClaimTypeReferenceId="displayName" />
      <OutputClaim ClaimTypeReferenceId="givenName" />

      <OutputClaim ClaimTypeReferenceId="surName" />

      <OutputClaim ClaimTypeReferenceId="extension_storeMembershipNumber" PartnerClaimType="storeMembershipNumber" />
   </OutputClaims>
   …
</TechnicalProfile>
```

10. Save the XML file and upload it to your B2C tenant as before.Now open the *SignupOrSignin.xml* policy XML file in the same folder.
12. Scroll down to the **RelyingParty** section of the custom policy. This XML element determines the interaction between Azure AD B2C and the consumer application that's making the request.
13. Add a new *OutputClaim* XML element with the *ClaimTypeReferenceId* attribute set to the *Id* attribute of the new claim type you added in the *B2C_1A_TrustFrameworkBase.xml* policy XML file in above steps 4 to 5. This allows to send the attribute to the consumer application in the token
14. Save the XML file and upload it to your B2C tenant as before.
15. Eventually run the custom policy using the Azure AD B2C blade in the Azure portal to test the new attribute being collected from the user during local account creation and sent in the token.

Applying a claims transformation to create a new claim

To continue our series of illustration in terms of user journey's customizations, we will then create a new claims transformation to create a new **displayName** claim from the **givenName** and **surname** claims and send it in the token to the consumer application.

This twofold process consists in:

1. Creating the claims transformation as such in the policy XML file.
2. Referencing the claims transformation from a claims provider to apply it.

As such, this section illustrates:

- How claims transformations are declared in the policy XML file, the role of the *InputClaims*, *OutputClaims* and *InputParameters* collections/sections.
- The relationship between claims transformation and claims providers.

As an illustration of the above process, proceed with the following steps:

1. Navigate this time to the folder in the "Starter Pack".
2. Open the base policy file, i.e. the *TrustFrameworkBase.xml* file, using an XML editor of your choice, for instance Visual Studio (Code).
3. Scroll down and locate the *ClaimsTransformations* section.
4. Add a new claims transformation with the following *ClaimsTransformation* XML element.

```
<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>
```

This is just the declaration of the claims transformation. The *Id* attribute is used to later reference the transformation. This claims transformation uses a predefined transformation method (e.g. *TransformationMethod* attribute), provides the input claims (e.g. *InputClaims* section) and input parameter (e.g. *InputParameters* section), and returns output claims (e.g. *OutputClaims* section).

5. In the **LocalAccountSignUpWithLogonEmail** technical profile, add the following **OutputClaimsTransformation**:

```
<OutputClaimsTransformations>
  <OutputClaimsTransformation ReferenceId="CreateDisplayNameFromGivenNameAndSurname"/>
</OutputClaimsTransformations>
```

Now when the technical profile will be invoked, and claims are returned from that technical profile, this claims transformation will be applied to create (or overwrite) the displayName claim.

6. Save the XML file and upload it to your B2C tenant as before.
7. Execute the **B2C_1A_signup_signin** policy. **displayName** should appear in the token that is issued to the consumer application.

Customizing the UI of a user journey

Azure AD B2C allows you to customize the look-and-feel of user experience on the various pages that can be potentially served and displayed by Azure AD B2C via your custom policies.

UI customization for a relating seamless user experience is key for any business-to-consumer solution. By seamless user experience, we mean an experience, whether on device or browser, where a user's journey through our service cannot be distinguished from that of the customer service they are using. (Localization is also part of this story to smoothly adapt the UI to the end-user.)

If you're not interested in this use case, you can skip this entire section and jump to the next one.

Understanding the CORS way for UI customization

The Identity Experience Engine in Azure AD B2C runs code in your consumer's browser and uses the modern and standard approach Cross-Origin Resource Sharing (CORS)[43] to load custom content from a specific URL that you specify in a custom policy to point to your HTML5/CSS templates (see next section): **"**Cross-origin resource sharing (CORS) is a mechanism that allows restricted resources (e.g. fonts) on a web page to be requested from another domain outside the domain from which the resource originated."[44]

Compared to the old traditional way, where template pages are owned by the solution where you provided limited text and images, where limited control of layout and feel was offered leading to more than difficulties to achieve a seamless experience, the CORS way supports HTML5 and CSS and allows you to:

- Host the content and the solution injects its controls using client-side script.
- Have full control over every pixel of layout and feel.

You can provide as many content pages as you like by crafting HTML5/CSS files as appropriate.

> **Note** For security reasons, the use of JavaScript is currently blocked for customization. It's going to be relax in a near future.

In each of your HTML5/CSS templates, you provide an "anchor" element, which corresponds to the required element in the HTML or the content page as illustrate hereafter. Azure AD B2C indeed requires that all content pages have this specific div.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Your page content's title!</title>
  </head>
  <body>

    <div id="api"></div>

  </body>
</html>
```

Azure AD B2C-related content for the page will be injected into this div, while the rest of the page is yours to control. The Azure AD B2C's JavaScript code pulls in your content and injects our HTML into this specific div element. Azure AD B2C injects the following controls as appropriate: account chooser control, login controls, multi-factor (currently phone-based) controls, and attribute collection controls. In terms of commitment, we will ensure that i) all our controls are HTML5 compliant and accessible, ii) all our controls can be fully styled, and iii) a control version will not regress.

---

[43] CROSS-ORIGIN RESOURCE SHARING W3C RECOMMENDATION 16 JANUARY 2014: http://www.w3.org/TR/cors/

[44] CROSS-ORIGIN RESOURCE SHARING: https://en.wikipedia.org/wiki/Cross-origin_resource_sharing

The merged content is eventually displayed as the dynamic document to your external user.

To ensure of the above works as expected, you must:

- Ensure your content is HTML5 compliant and accessible.
- Ensure your content server is enabled for CORS.

> **Note** To verify that the site you are hosting your content on has CORS enabled and test CORS requests, you can use the site http://test-cors.org/. Thanks to this site, you can simply either send the CORS request to a remote server (to test if CORS is supported) or send the CORS request to a test server (to explore certain features of CORS).

> **Note** The site http://enable-cors.org/ also constitutes a more than useful resources on CORS.

- Serve content over HTTPS.
- Use absolute URLS such as *https://yourdomain/content* for all links and CSS content.

Thank to this CORS-based approach, the end users will then have consistent experiences between your application and the pages served by Azure AD B2C.

Adding static UI customization to a user journey

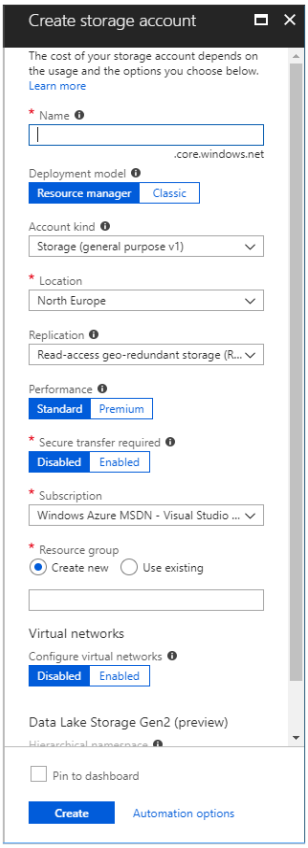
> **Note** For more information, see article AZURE ACTIVE DIRECTORY B2C: CONFIGURE UI CUSTOMIZATION IN A CUSTOM POLICY[45].

Creating a storage account for your HTML5/CSS templates

To create a storage account, proceed with the following steps:

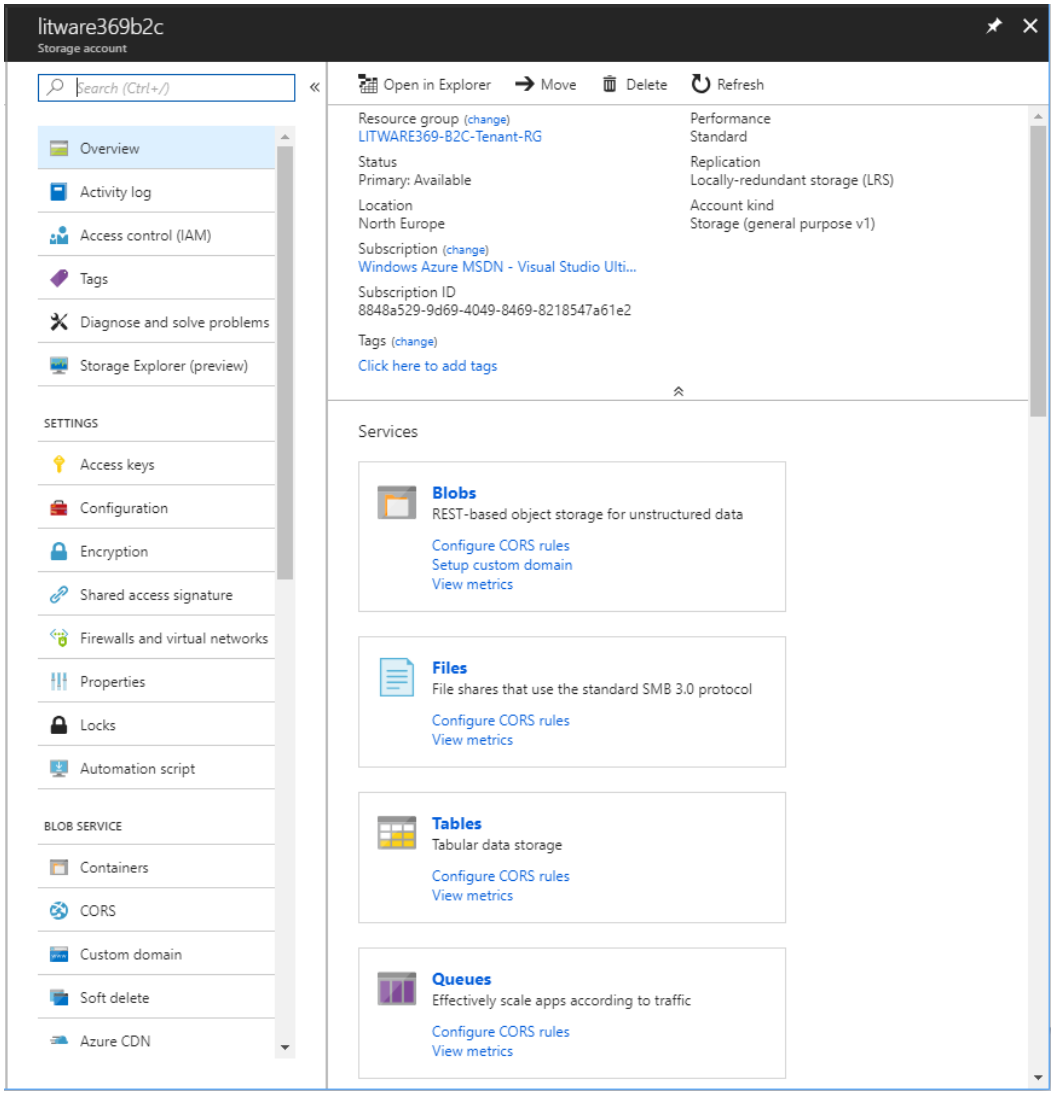
1. Open a browsing session and navigate to the Azure portal at https://portal.azure.com.
2. Sign in with your administrative credentials.
3. Click **+ New** > **Data + Storage** > **Storage account**. A **Create storage account** blade opens.

[45] AZURE ACTIVE DIRECTORY B2C: CONFIGURE UI CUSTOMIZATION IN A CUSTOM POLICY: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-ui-customization-custom

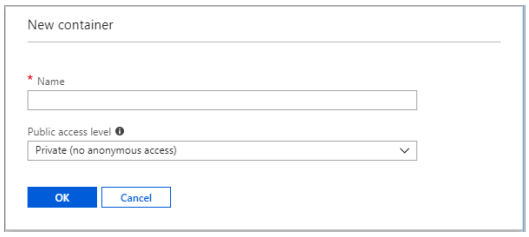


Note You will need an Azure subscription to create an Azure Blob Storage account. You can sign up a free trial at the Azure website at https://azure.microsoft.com/en-us/free/?v=18.23.

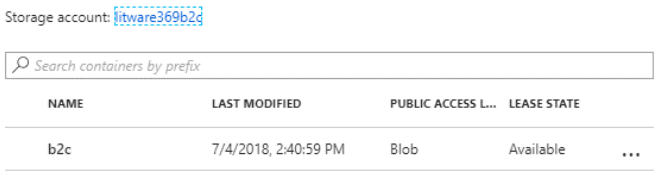
4. In **Name**, provide a name for the storage account, for example, "*litware369b2c*" in our configuration. This value will be later referred as to *storageAccountName*.
5. Pick the appropriate selections for the location, the replication mode, the performance/pricing tier, the resource group and the subscription. Make sure that you have the **Pin to Startboard** option checked. Click **Create**.
6. Go back to the Startboard and click the storage account that you just created.



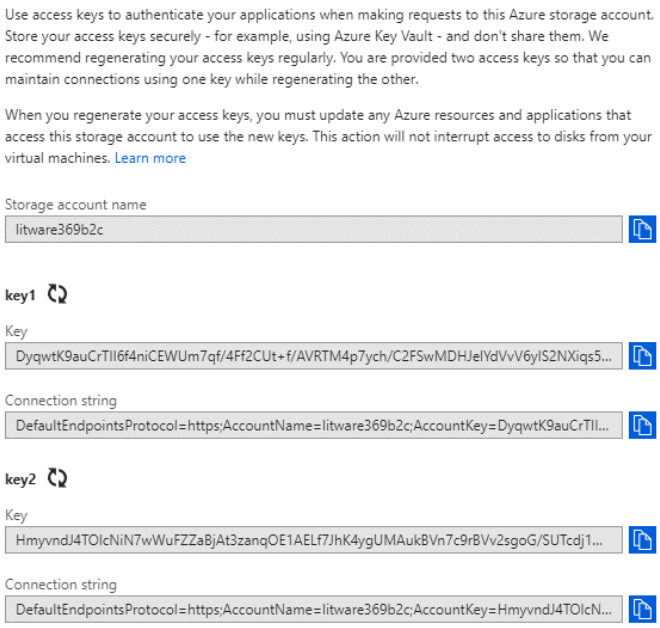
7. In the **Services** section, click **Blobs**. A **Blob service** blade opens up.
8. Click **+ Container**.



9. In **Name**, provide a name for the container. For example, in our configuration, "b2c". This value will be later referred as to *containerName*.
10. Select **Blob (anonymous read access for blobs only)** as the **Access type**. Click **OK**.
11. The container that you created will appear in the list on the **Containers** blade.



12. Close the blade.
13. On the storage account blade, click the Key icon. An **Access keys** blade opens up.



14. Write down the value of the key under **key1**. This value will be later referred as *key1*.

Important note	key1 is an important security credential.

Downloading the helper tool

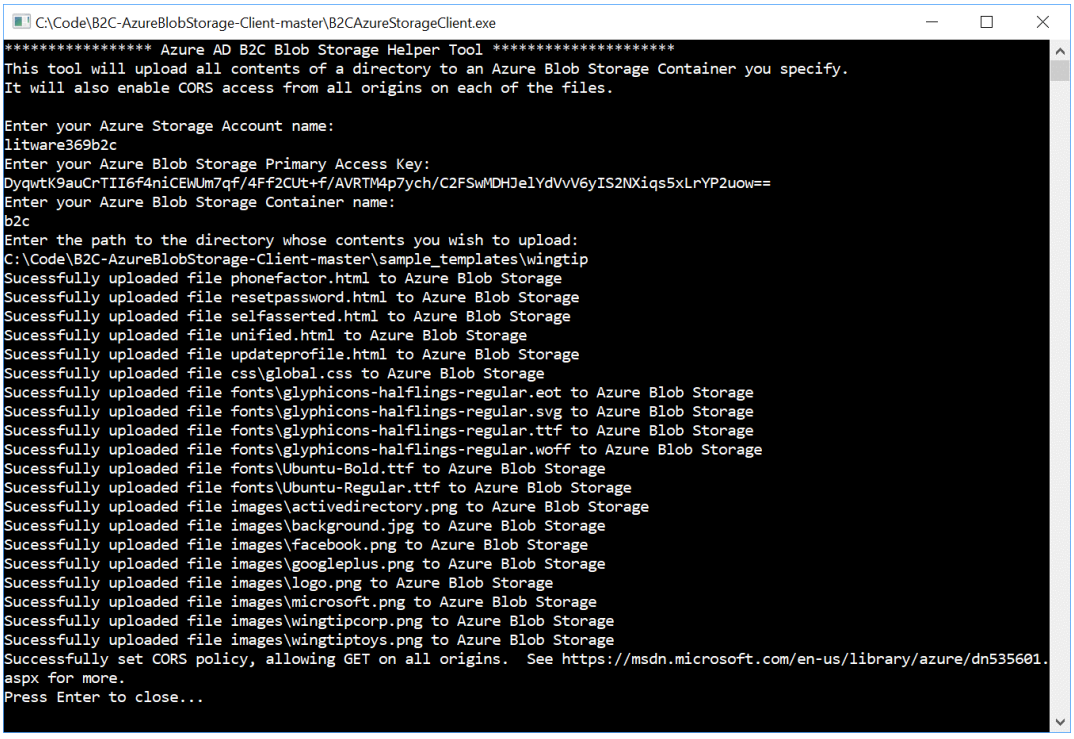
To download the so-called Azure Blob Storage helper tool, proceed with the following step:

1. Download the helper tool as a .zip file from GitHub at https://github.com/azureadquickstarts/b2c-azureblobstorage-client/archive/master.zip.
2. Save the *B2C-AzureBlobStorage-Client-master.zip* file on your local machine.
3. Extract the content of the *B2C-AzureBlobStorage-Client-master.zip* file on your local disk, for example under the **Starter-Pack** folder. This will create a *B2C-AzureBlobStorage-Client-master* folder underneath.
4. Open that folder and extract the content of the archive file *B2CAzureStorageClient.zip* within it.

Uploading the sample files

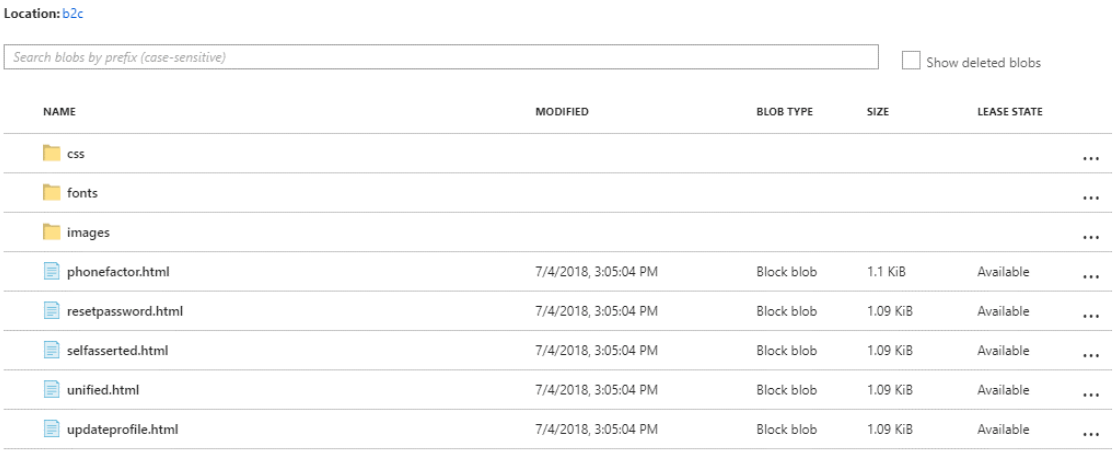
To upload the sample files from the content pack, proceed with the following steps:

1. From the Windows Explorer, navigate to the above folder *B2C-AzureBlobStorage-Client-master* located under the **Starter-Pack** folder.
2. Run the *B2CAzureStorageClient.exe* file within. This program will simply upload all the files in the directory that you specify to your storage account and enable CORS access for those files.
3. When prompted, specify:
    a. The name of your storage account, i.e. *storageAccountName*, for example *litware369b2c* in our configuration.
    b. The primary access key of your azure blob storage, i.e. *key1*, for example *litware369b2c* in our configuration.
    c. The name of your storage blob storage container, i.e. *containerName*, for example *b2c* in our configuration.
    d. And eventually, for the sake of this illustration, the path of the Wingtip Toys sample files, e.g. the *B2C-AzureBlobStorage-Client-master\sample_templates\wingtip* folder under the **Starter-Pack** folder.



If you followed the steps above, the HTML5 and CSS files for the fictitious company Wingtip Toys will now be pointing to your storage account.

You can verify that the content has been uploaded correctly by opening the related container blade in the Azure portal.



You can alternatively verify that the content has been uploaded correctly with the Microsoft Azure Storage Explorer tool[46] or by trying to access on the browser. The page should be displayed in the browser.

> **Note** For additional information, see the article AZURE ACTIVE DIRECTORY B2C: A HELPER TOOL USED TO DEMONSTRATE THE PAGE USER INTERFACE (UI) CUSTOMIZATION FEATURE[47].

The following table describes the purpose of the above HTML5 pages.

HTML5 template	Description
phonefactor.html	This page can be used as a template for a multi-factor authentication page. (See below)
resetpassword.html	This page can be used as a template for a forgot password page. (See below)
selfasserted.html	This page can be used as a template for a social account sign-up page, a local account sign-up page, or a local account sign-in page. (See below)
unified.html	This page can be used as a template for a unified sign-up or sign-in page. (See below)
updateprofile.html	This page can be used as a template for a profile update page. (See below)

Ensuring the storage you are hosting your HTML5/CSS templates from has CORS enabled

As stated before, CORS (Cross-Origin Resource Sharing) must be enabled on your endpoint for Azure AD B2C to load your content. This is because your content is hosted on a different domain than the domain the Identity Experience Framework in Azure AD B2C will be serving the page from.

To verify that the storage you are hosting your content on has CORS enabled, proceed with the following steps:

1. Open a browsing session and navigate to the page *unified.html* using the full URL of its location in your storage account:

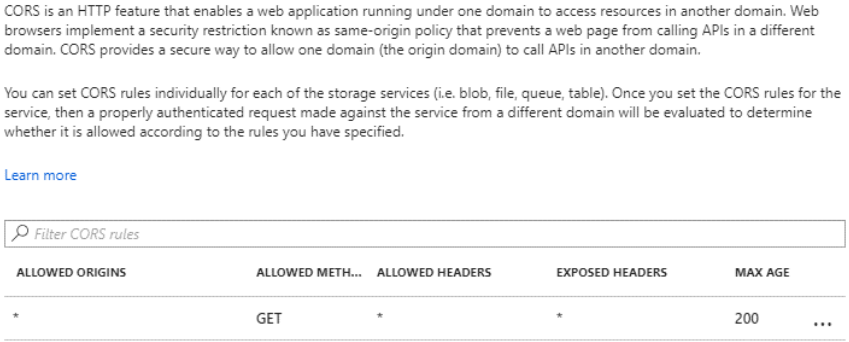
[46] Microsoft Azure Storage Explorer tool: https://azure.microsoft.com/en-us/features/storage-explorer/
[47] AZURE ACTIVE DIRECTORY B2C: A HELPER TOOL USED TO DEMONSTRATE THE PAGE USER INTERFACE (UI) CUSTOMIZATION FEATURE: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-reference-ui-customization-helper-tool

https://<*your_storageAccount*>.blob.core.windows.net/<*your_containerName*>/unified.html

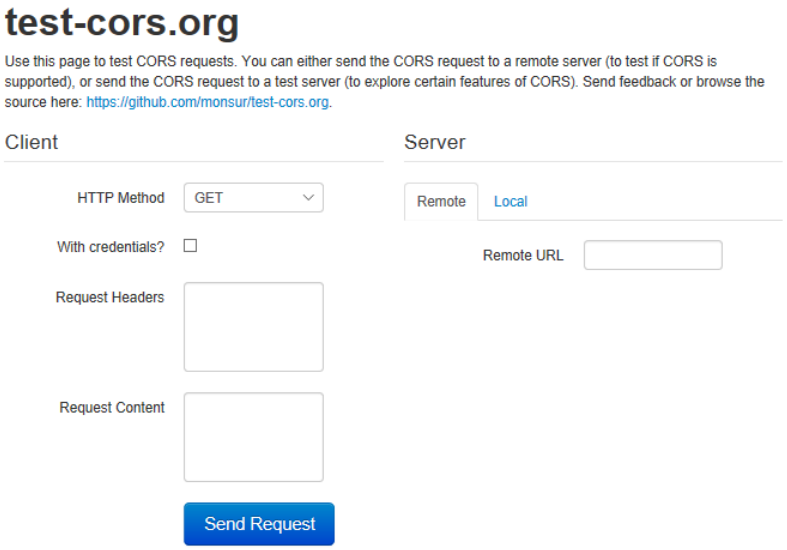
For example, in our configuration:

https://litware369b2c.blob.core.windows.net/b2c/unified.html

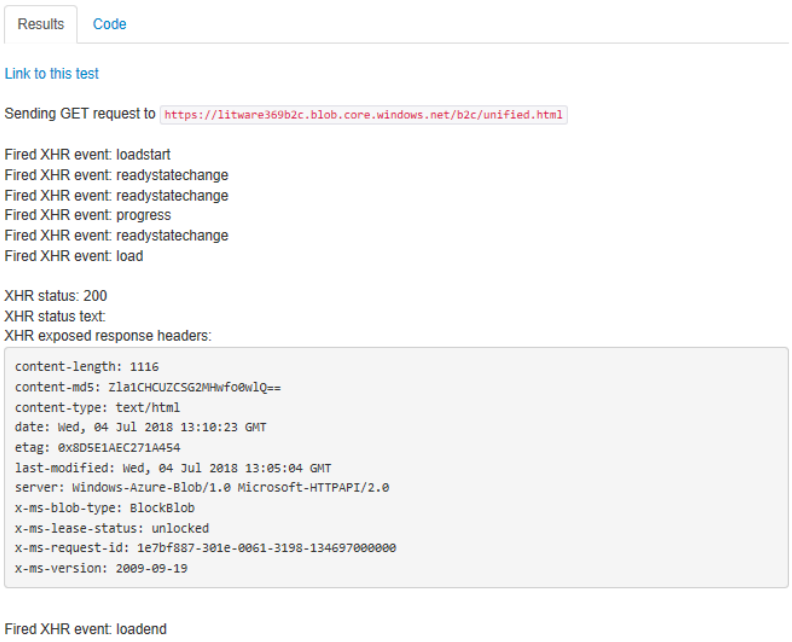
2. Navigate to your storage account in the Azure portal, select CORS under BLOB SERVICE. You can view the CORS rule that has been set by the helper tool for your container.



3. Now navigate to http://test-cors.org. This site allows you to verify that the page you are using has CORS enabled.



4. In **Remote URL**, enter the full URL for your *unified.html* content, and click **Send Request**.
5. Verify that the output in the **Results** section contains "XHR status: 200". This indicates that CORS is enabled.



Adding a link to your HTML5/CSS templates to your user journey

This section describes how add a link to your custom HTML5/CSS templates to your user journey by editing a custom policy directly.

> **Note** For details that pertain to how to configure and use a storage for your custom HTML5/CSS templates, see section § *Managing your HTML5/CSS templates for the policies* of the third document of this series.

The custom HTML5/CSS templates to use in your user journey have to be specified in a list of content definitions that can be used in those user journeys. For that purpose, an optional *ContentDefinitions* XML element must be declared under the *BuildingBlocks* section of your custom policy file.

The *ContentDefinitions* section can contains a series of *ContentDefinition* XML elements. The *Id* attribute of this element allows to specify the type of pages that relates to the content definition, and thus the context in which a custom HTML5/CSS template is going to be used. The following table describes the set of the *id* attributes.

Id	Description
api.error	**Error page**. This page is displayed when an exception or an error is encountered.
api.idpselections	**Identity provider selection page**. This page contains a list of identity providers that the user can choose from during sign-in. These are either enterprise identity providers, social identity providers such as Facebook and Google+, or local accounts (based on email address or user name).
api.idpselections.signup	**Identity provider selection for sign-up**. This page contains a list of identity providers that the user can choose from during sign-up. These are either enterprise identity providers, social identity providers such as Facebook and Google+, or local accounts (based on email address or user name).
api.localaccountpasswordreset	**Forgot password page**. This page contains a form that the user has to fill to initiate their password reset.

api.localaccountsignin	**Local account sign-in page**. This page contains a sign-in form that the user has to fill in when signing in with a local account that is based on an email address or a user name. The form can contain a text input box and password entry box.
api.localaccountsignup	**Local account sign-up page**. This page contains a sign-up form that the user has to fill in when signing up for a local account that is based on an email address or a user name. The form can contain different input controls such as text input box, password entry box, radio button, single-select drop-down boxes, and multi-select check boxes.
api.phonefactor	**Multi-factor authentication page**. On this page, users can verify their phone numbers (using text or voice) during sign-up or sign-in
api.selfasserted	**Social account sign-up page**. This page contains a sign-up form that the user has to fill in when signing up using an existing account from a social identity provider such as Facebook or Google+. This page is similar to the above social account sign-up page with the exception of the password entry fields.
api.selfasserted.profileupdate	**Profile update page**. This page contains a form that the user can use to update their profile. This page is similar to the above social account sign-up page with the exception of the password entry fields.
api.signuporsignin	**Unified sign-up or sign-in (SUSI) page**. This page handles both sign-up & sign-in of users, who can use enterprise identity providers, social identity providers such as Facebook or Google+, or local accounts.
api.signuporsigninwithkmsi	**Unified sign-up or sign-in (SUSI) page with 'Keep me signed in (KMSI)' capability**. This page is similar to the above unified sign-up or sign-in (SUSI) page plus the KMSI capability. For more information, see article AZURE ACTIVE DIRECTORY B2C: ENABLE 'KEEP ME SIGNED IN (KMSI)'[48].

Each *ContentDefinition* XML element contain in turn an inner *LoadUri* XML element that points to the intended HTML5/CSS template.

> **Note** For additional detail on the content definitions, see eponym section § *Specifying the content definitions* of the sixth document of this series.

To configure a custom HTML5/CSS template to use in your user journey, proceed with the following steps:

1. Navigate this time to the folder in the "Starter Pack".
2. Open the base policy file, i.e. the *TrustFrameworkBase.xml* file, using an XML editor of your choice, for instance Visual Studio (Code).
3. Search for the *ContentDefinitions* element.
4. For each *ContentDefinition* b underneath, modify the *LoadUri* XML element to specify the URL of your custom HTML5/CSS template.

   For example, search for the *ContentDefinition* node that contains Id="api.signuporsignin" and Change the value of LoadUri from ~/tenant/default/unified to https://litware369b2c.blob.core.windows.net/b2c/unified.html.

[48] AZURE ACTIVE DIRECTORY B2C: ENABLE 'KEEP ME SIGNED IN (KMSI)': https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-reference-kmsi-custom

```
<ContentDefinition Id="api.signuporsignin">
    <LoadUri>https://litware369b2c.blob.core.windows.net/b2c/unified.html</LoadUri>
    <RecoveryUri>~/common/default_page_error.html</RecoveryUri>
    <DataUri>urn:com:microsoft:aad:b2c:elements:unifiedssp:1.0.0</DataUri>
    <Metadata>
       <Item Key="DisplayName">Signin and Signup</Item>
    </Metadata>
</ContentDefinition>
```

5. Repeat above step 4 to set additional content definition as needed for your user journey.
6. Save the XML file and upload it to your B2C tenant as before.
7. Test your user journey as usual. The custom HTML5/CSS template(s) should now be used by Azure AD B2C.

Adding dynamic UI customization to a user journey

In the previous section, you upload HTML5/CSS templates to an Azure Blob storage. Those HTML5 HTML5/CSS templates are static and render the same HTML content for each request.

With custom policy, you can also customize the look and feel for your user dynamically.

> **Note** For more information, see article AZURE ACTIVE DIRECTORY B2C: CONFIGURE THE UI WITH DYNAMIC CONTENT BY USING CUSTOM POLICIES[49].

If you're not interested in this use case, you can skip this entire section and jump to the next one.

Custom policies indeed allow you to send through parameters in a query string. These parameters pass on to your HTML endpoint and can dynamically change the page content. For example, as a very basic illustration, you can change the B2C sign-up or sign in background image, based on a parameter you pass from your web/mobile application. Such a capability can be used to sustain a specific marketing campaign.

For the sake of the illustration, we leverage here the sample policy files under the folder *scenarios\ aadb2c-ief-ui-customization* of the "Starter Pack" along with the provided code sample. This sample code is an ASP.NET Core web app, which can accept query string parameters and respond accordingly by serving the appropriate HTML5/CSS template. This template is customized in accordance to the received query string parameters.

To do so, you will have to:

- Build the ASP.NET Core sample web app. This MVC based sample web app hosts your HTML5/CSS templates.
- Publish this web app to Azure App Service.
- Set cross-origin resource sharing (CORS) for this web app.
- Update the custom policies to uverride the *LoadUri* elements to point to the intended view of this MVC based web app.
- Upload the custom policies.
- Test the custom policies by using **Run Now**.

[49] AZURE ACTIVE DIRECTORY B2C: CONFIGURE THE UI WITH DYNAMIC CONTENT BY USING CUSTOM POLICIES: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-ui-customization-custom-dynamic

The next section detail each of the above steps.

Building the code sample

Follow the instructions as per section § *Building the Contoso.AADB2C.UI* code sample in the Appendix.

Ensuring the HTML5/CSS templates from the web app have CORS enabled

As stated before, CORS (Cross-Origin Resource Sharing) must be enabled on your endpoint for Azure AD B2C to load your content. This is because your content is hosted on a different domain than the domain the Identity Experience Framework in Azure AD B2C will be serving the page from.

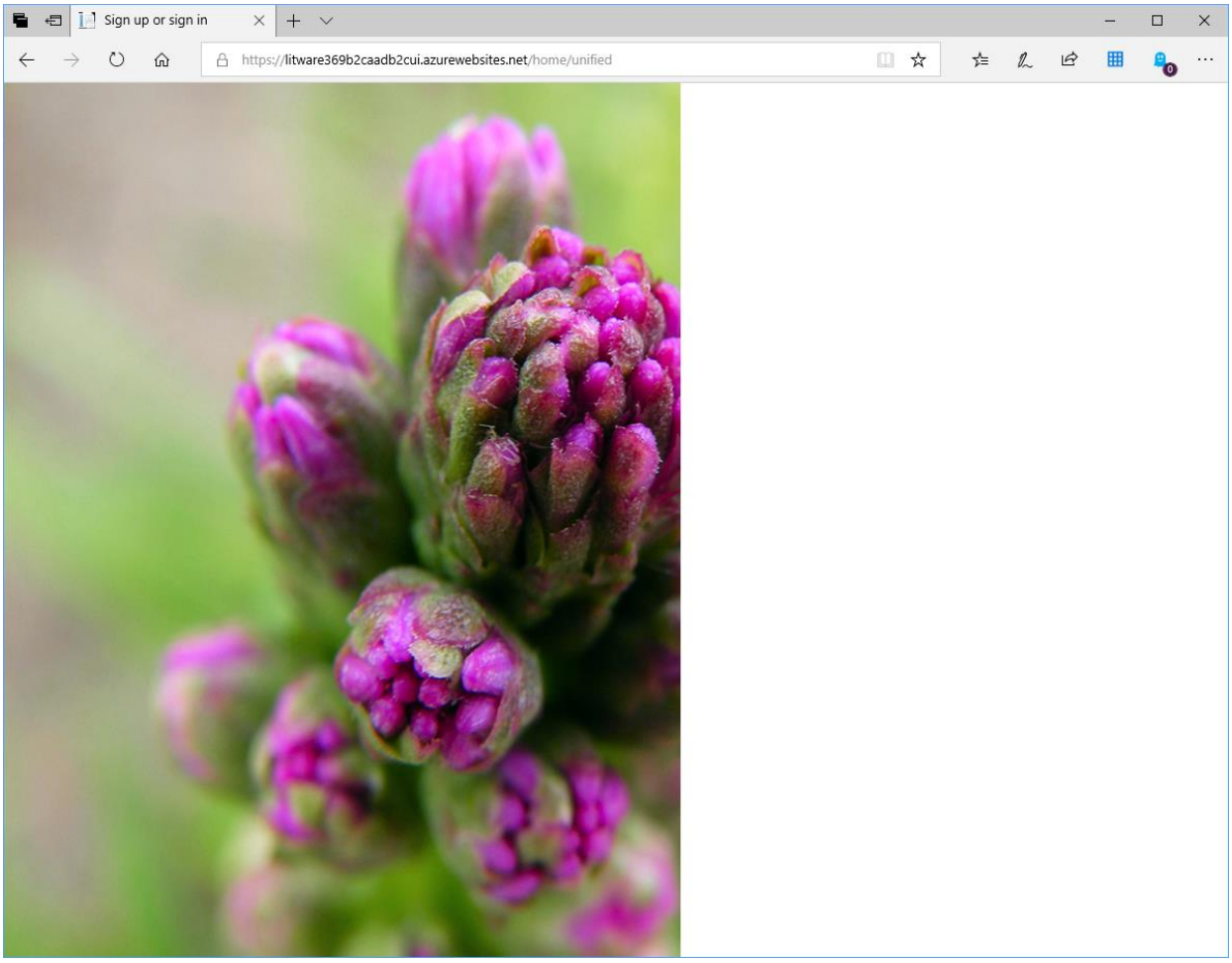
To verify that the web app you are hosting your content on has CORS enabled, proceed with the following steps:

1. Open a browsing session and navigate to the *unified* view using the URL of the published web app:

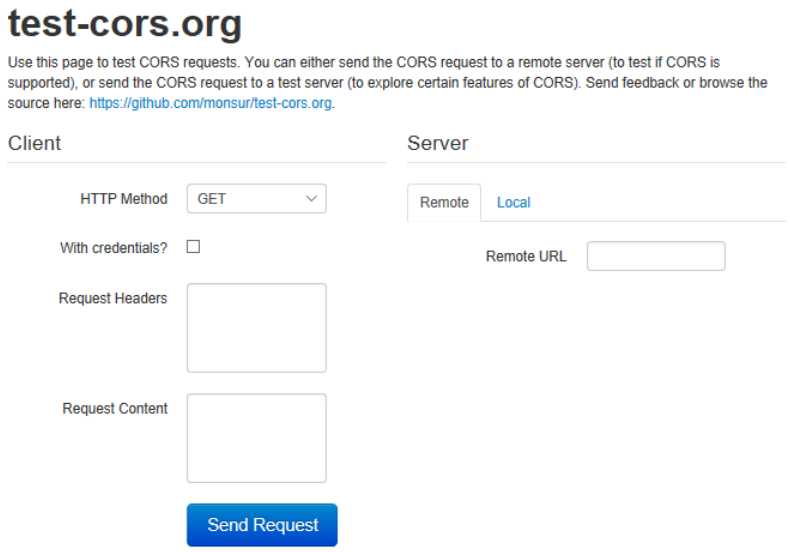
   https://<*your_app_name*>.azurewebsites.net/home/unified

   For example, in our configuration:

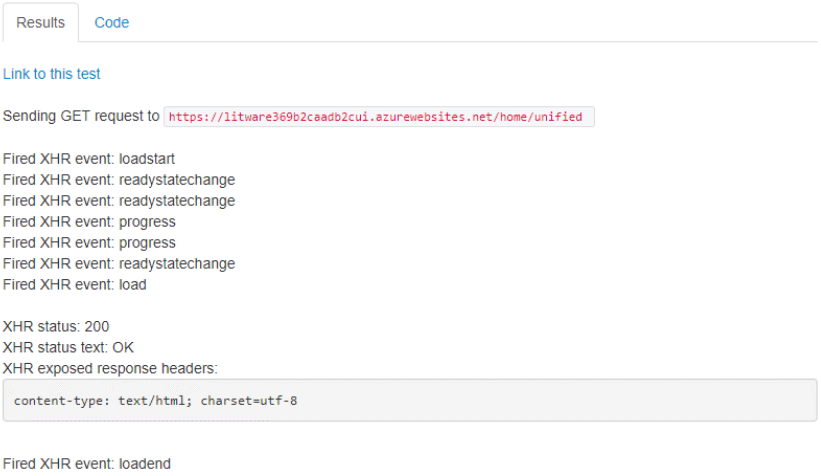
   https://litware369b2caadb2cui.azurewebsites.net/home/unified



2. Now navigate to http://test-cors.org. This site allows you to verify that the page you are using has CORS enabled.



3. In **Remote URL**, enter the full URL for your *unified* view, and click **Send Request**.
4. Verify that the output in the **Results** section contains "XHR status: 200". This indicates that CORS is enabled. In other words, your content web app is enabled for CORS



At this stage, the HTML5/CSS templates are ready to use. However, they are not available in the *ContentDefinition* element of your custom policy.

Updating the extension custom policy

Configuring the custom HTML5/CSS template(s) to use

To configure the custom HTML5/CSS template(s) to use in your user journeys, proceed with the following steps:

1. Navigate this time to the folder in the "Starter Pack".

2. Open the base policy file, i.e. the *TrustFrameworkBase.xml* file, using an XML editor of your choice, for instance Visual Studio (Code).
3. Search for the *ContentDefinition* element, and then copy the entire content of the *ContentDefinitions* node.

```
<ContentDefinitions>

   <!-- This content definition is to render an error page that displays unhandled errors. -->
   <ContentDefinition Id="api.error">
      <LoadUri>~/tenant/default/exception.cshtml</LoadUri>
      <RecoveryUri>~/common/default_page_error.html</RecoveryUri>
      <DataUri>urn:com:microsoft:aad:b2c:elements:globalexception:1.1.0</DataUri>
      <Metadata>
         <Item Key="DisplayName">Error page</Item>
      </Metadata>
   </ContentDefinition>

   <ContentDefinition Id="api.idpselections">
      <LoadUri>~/tenant/default/idpSelector.cshtml</LoadUri>
      <RecoveryUri>~/common/default_page_error.html</RecoveryUri>
      <DataUri>urn:com:microsoft:aad:b2c:elements:idpselection:1.0.0</DataUri>
      <Metadata>
         <Item Key="DisplayName">Idp selection page</Item>
         <Item Key="language.intro">Sign in</Item>
      </Metadata>
   </ContentDefinition>

   <ContentDefinition Id="api.idpselections.signup">
      <LoadUri>~/tenant/default/idpSelector.cshtml</LoadUri>
      <RecoveryUri>~/common/default_page_error.html</RecoveryUri>
      <DataUri>urn:com:microsoft:aad:b2c:elements:idpselection:1.0.0</DataUri>
      <Metadata>
         <Item Key="DisplayName">Idp selection page</Item>
         <Item Key="language.intro">Sign up</Item>
      </Metadata>
   </ContentDefinition>

   <ContentDefinition Id="api.signuporsignin">
      <LoadUri>~/tenant/default/unified.cshtml</LoadUri>
      <RecoveryUri>~/common/default_page_error.html</RecoveryUri>
      <DataUri>urn:com:microsoft:aad:b2c:elements:unifiedssp:1.0.0</DataUri>
      <Metadata>
         <Item Key="DisplayName">Signin and Signup</Item>
      </Metadata>
   </ContentDefinition>

   <ContentDefinition Id="api.selfasserted">
      <LoadUri>~/tenant/default/selfAsserted.cshtml</LoadUri>
      <RecoveryUri>~/common/default_page_error.html</RecoveryUri>
      <DataUri>urn:com:microsoft:aad:b2c:elements:selfasserted:1.1.0</DataUri>
      <Metadata>
         <Item Key="DisplayName">Collect information from user page</Item>
      </Metadata>
   </ContentDefinition>

   <ContentDefinition Id="api.selfasserted.profileupdate">
      <LoadUri>~/tenant/default/updateProfile.cshtml</LoadUri>
      <RecoveryUri>~/common/default_page_error.html</RecoveryUri>
      <DataUri>urn:com:microsoft:aad:b2c:elements:selfasserted:1.1.0</DataUri>
      <Metadata>
         <Item Key="DisplayName">Collect information from user page</Item>
      </Metadata>
   </ContentDefinition>

   <ContentDefinition Id="api.localaccountsignup">
      <LoadUri>~/tenant/default/selfAsserted.cshtml</LoadUri>
      <RecoveryUri>~/common/default_page_error.html</RecoveryUri>
      <DataUri>urn:com:microsoft:aad:b2c:elements:selfasserted:1.1.0</DataUri>
      <Metadata>
         <Item Key="DisplayName">Local account sign up page</Item>
      </Metadata>
   </ContentDefinition>

   <ContentDefinition Id="api.localaccountpasswordreset">
      <LoadUri>~/tenant/default/selfAsserted.cshtml</LoadUri>
```

```
      <RecoveryUri>~/common/default_page_error.html</RecoveryUri>
      <DataUri>urn:com:microsoft:aad:b2c:elements:selfasserted:1.1.0</DataUri>
      <Metadata>
         <Item Key="DisplayName">Local account change password page</Item>
      </Metadata>
   </ContentDefinition>

</ContentDefinitions>
```

4. Now open the extension policy file, i.e. the *TrustFrameworkExtensions.xml* file and then search for the *BuildingBlocks* element.
5. Paste the entire contents of the *ContentDefinitions* node that you copied as a child of the *BuildingBlocks* element.
6. Search for the *ContentDefinition* node that contains Id="api.signuporsignin" in the XML that you copied.
7. Change the value of LoadUri from ~/tenant/default/unified to https://<*your_app_name*>.azurewebsites.net/home/unified.

   The related XML should look like the following:

```
<ContentDefinition Id="api.signuporsignin">

   <LoadUri>https://litware369b2caadb2cui.azurewebsites.net/home/unified</LoadUri>

   <RecoveryUri>~/common/default_page_error.html</RecoveryUri>
   <DataUri>urn:com:microsoft:aad:b2c:elements:unifiedssp:1.0.0</DataUri>
   <Metadata>
      <Item Key="DisplayName">Signin and Signup</Item>
   </Metadata>
</ContentDefinition>
```

8. Save the XML file and upload it to your B2C tenant as before.
9. Test your user journey as usual by using **Run Now**. You should be able to see your custom HTML5 with the background that you created earlier.

Adding dynamic content

The background should now change based on a query string parameter named campaignId. Your relying party (RP) application (web and mobile apps) sends the parameter to the Identity Experience Framework in Azure AD B2C. The relying party (RP) custom policy reads the parameter and sends its value to your HTML5/CSS template, i.e. your MVC based ASP.NET Core web app.

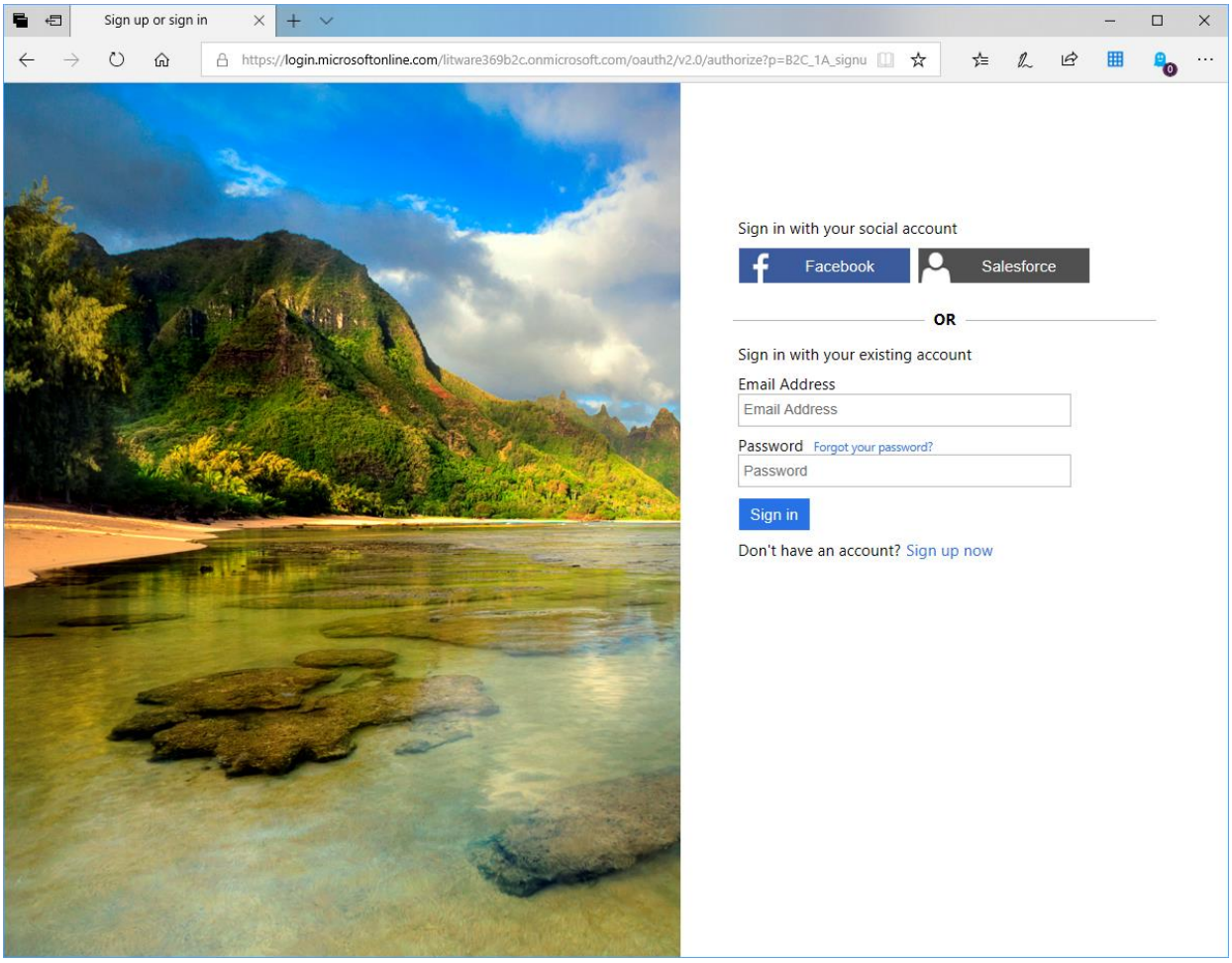
Proceed with the following steps:

1. From the folder in the "Starter Pack", open the Sign-up or Sign-in (SUSI) policy file, i.e. the *SignUpOrSignin.xml* file, using an XML editor of your choice, for instance Visual Studio (Code).
2. Search for the *DefaultUserJourney* node and right after the *DefaultUserJourney* node, add the following XML snippet.

```
<UserJourneyBehaviors>
    <ContentDefinitionParameters>
        <Parameter Name="campaignId">{OAUTH-KV:campaignId}</Parameter>
    </ContentDefinitionParameters>
</UserJourneyBehaviors>
```

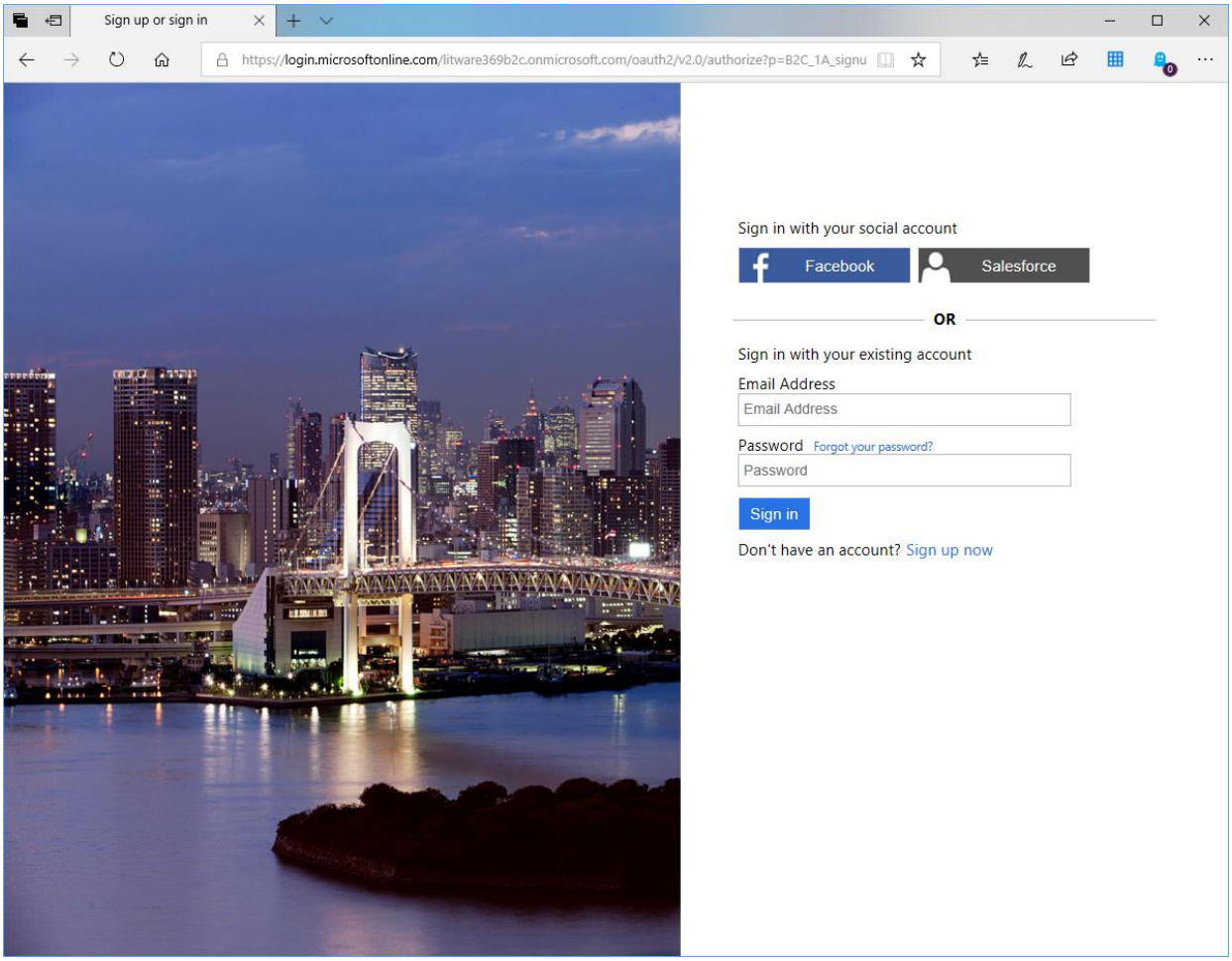
3. Save the XML file and upload it to your B2C tenant as before.
4. Test the **B2C_1A_signup_signin** policy as usual by using **Run Now**. You should see the same background image that was previously displayed.
5. Copy the URL from the browser's address bar and add the *campaignId* query string parameter to the URI. For example, add &campaignId=hawaii, as shown in following image:

   https://login.microsoftonline.com/litware369b2c.onmicrosoft.com/oauth2/v2.0/authorize?p=B2C_1A_signup_signin&client_id=95b7953b-a499-48b9-bc1b-48bb095d6939&nonce=defaultNonce&redirect_uri=https%3A%2F%2Fjwt.ms&scope=openid&response_type=id_token&prompt=login&campaignId=hawaii



6. Change the value to “tokyo”, and then press ENTER.

   https://login.microsoftonline.com/litware369b2c.onmicrosoft.com/oauth2/v2.0/authorize?p=B2C_1A_signup_signin&client_id=95b7953b-a499-48b9-bc1b-48bb095d6939&nonce=defaultNonce&redirect_uri=https%3A%2F%2Fjwt.ms&scope=openid&response_type=id_token&prompt=login&campaignId=tokyo



7. Select the **Sign up now** link on the sign-in page. The browser displays the default background image, not the image you defined. This behavior arises because you've changed only the sign-up or sign-in page.

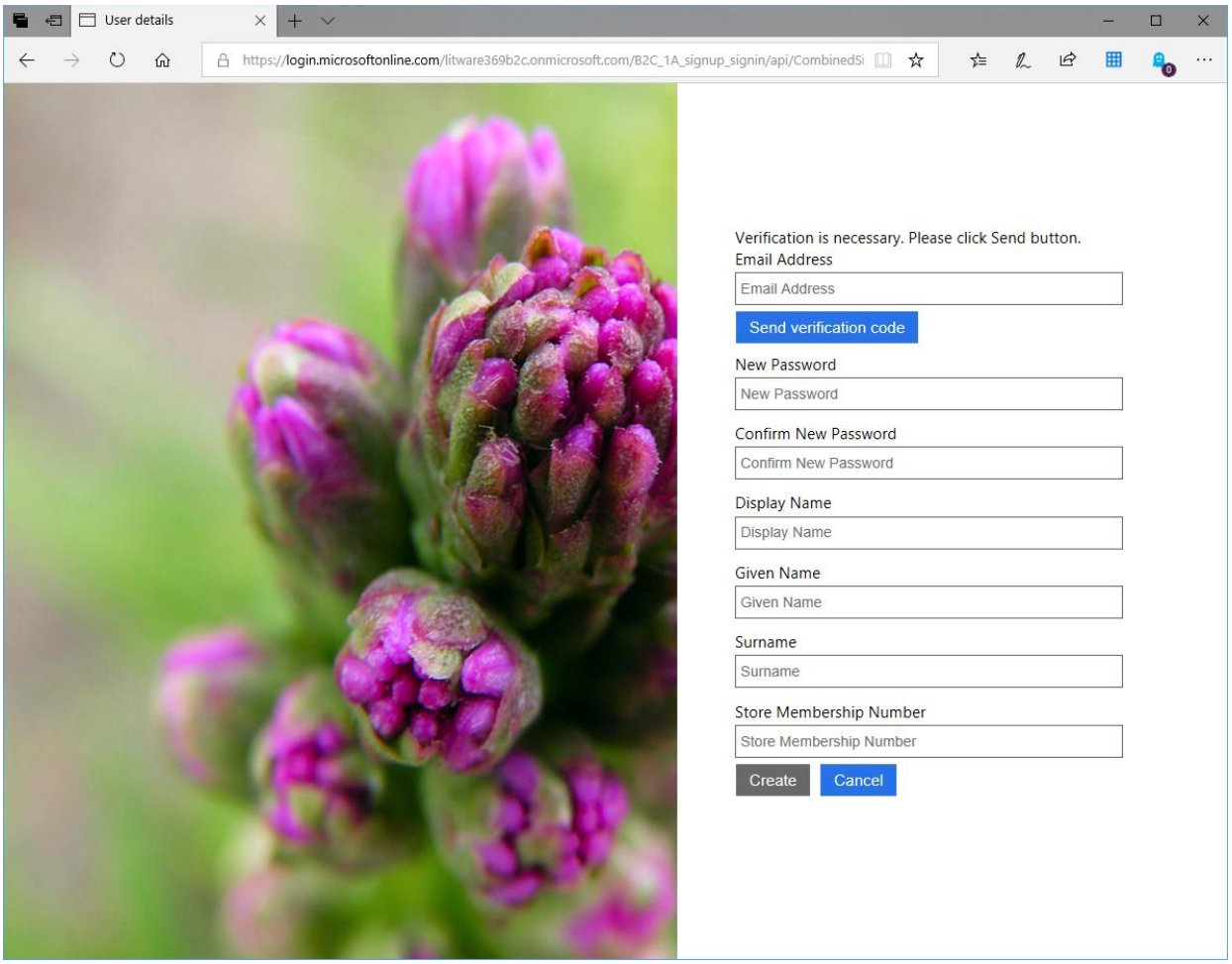
8. To change the rest of the Self-Assert content definitions, open the extension policy file, i.e. the *TrustFrameworkExtensions.xml* file.

9. In your extension policy, find the *ContentDefinition* nodes that contain Id="api.selfasserted", Id="api.localaccountsignup", and Id="api.localaccountpasswordreset", and set the LoadUri attribute to your selfasserted URI, i.e. https://<*your_app_name*>.azurewebsites.net/home/selfasserted.

   The related XML should look like the following for the *ContentDefinition* node that contain Id="api.selfasserted":

```
<ContentDefinition Id=" api.selfasserted">
   <LoadUri>https://litware369b2caadb2cui.azurewebsites.net/home/selfasserted</LoadUri>
   <RecoveryUri>~/common/default_page_error.html</RecoveryUri>
   <DataUri>urn:com:microsoft:aad:b2c:elements:selfasserted:1.1.0</DataUri>
   <Metadata>
      <Item Key="DisplayName">Collect information from user page</Item>
   </Metadata>
</ContentDefinition>
```

10. Save the XML file, upload it to your B2C tenant, and ensure that it passes validation.
11. Run the **B2C_1A_signup_signin** policy, and then select **Sign up now** to see the result.



Triggering language-specific content for the user journey

You can trigger language-specific content by just adding a query string parameter *ui_locales* to a standard request to the Identity Experience Framework in Azure AD B2C in order to render the end user experience in the target language.

For example, following is a request to run a Sign-Up or Sign-In (SUSI) policy in French:

https://login.microsoftonline.com/litware369b2c.onmicrosoft.com/oauth2/v2.0/authorize?p=B2C_1A_signup_signin&client_id=95b7953b-a499-48b9-bc1b-48bb095d6939&nonce=defaultNonce&redirect_uri=https%3A%2F%2Fjwt.ms&scope=openid&response_type=id_token&prompt=login&ui_locales=fr

As illustrated, when you call into the Identity Experience Framework in Azure AD B2C, the page is translated to the locale that you have indicated. This type of configuration gives you complete control over the languages in your user journey and ignores the language settings of the user's browser.

> **Important note** Custom policies must be updated to provide language-specific custom UI pages, and override any string on any page, as required, for the right end user experience. Same is true if you're using extensions properties, a.k.a. custom attributes. For more information, see article LANGUAGE CUSTOMIZATION IN CUSTOM POLICIES[50].

You might not need that level of control over what languages your customer sees. If you don't provide a *ui_locales* parameter, like we did so far, the user's experience is dictated by their browser's settings. You can still control which languages your user journey is translated to by adding it as a supported language. If a customer's browser is set to show a language that you don't want to support, then the language that you selected as a default in supported cultures is shown instead:

- *ui-locales* specified language: your user journey is translated to the language that's specified here. If the *ui_locales* parameter specifies one language that is not supported, the user journey is translated to the policy default language.
- Browser-requested language: If no *ui_locales* parameter was specified, your user journey is translated to the user agent (typically the browser) requested language, if the language is supported.
- Custom policy default language: If the browser doesn't specify a language, or it specifies one that is not supported, the user journey is translated to the policy default language. See section § *Specifying the supported locales* in the sixth document of this series.

[50] LANGUAGE CUSTOMIZATION IN CUSTOM POLICIES: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-language-customization-custom

Migrating users to your B2C tenant

To migrate work or school users from an identity provider database, a local directory, to your B2C tenant, you can use the Azure AD Graph API to retrieve the details of local users and groups, and then recreate the same structures in your B2C tenant. However, there are a number of issues that you should consider. This is the purpose of the next sections.

Understanding the primary considerations for the migration

Azure AD B2C allows you to migrate users through the Azure AD Graph API. As stated above, you simply read user's date from the local directory, and recreate a new account in your B2C tenant, but arguably, the most important concern here is the dependence on the user's password.

It may be possible to recreate users with their original passwords if you have access to them, but if the passwords are (salted and) stored in hash format, or encrypted, this approach might not be possible (or even desirable). Instead, you can opt to generate a random password and require users to reset their password the first time they log in after the migration.

From a practical standpoint, the above leads to five different migration flows:

1. Migrating users with existing password.
2. Migrating users without password and forcing them to change the password via a password reset policy.
3. Migrating users with password harvesting.
4. Pre-migrating users without password and setting password just-in-time
5. Migrating active users just-in-time.

The next sections illustrate each flow in order.

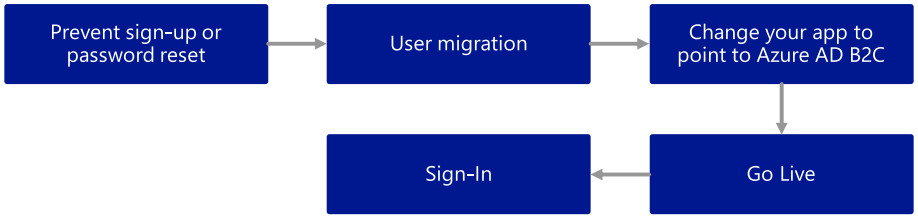
Migrating users with existing password

This first migration flow fits when you have access to user credentials (user name and password) in a clear text, or the credentials are encrypted, but you are able to decrypt them.

This first migration flow is based on a pre-migration process. This process basically implies:

- Reading the users' data from the old identity provider and create new accounts in your B2C tenant.
- Changing your mobile or web application(s) to authenticate against your B2C tenant by executing a sign-in (custom) policy.

Once the pre-migration process is completed, you can go live users are able to sign-in immediately.



This process can be applied to migrate all users or only pre-selected ones. Before going live, you should prevent users from updating their password or creating new ones.

Generally speaking, migration time can take long (depending on the number of users), but you can pre-migrate users a month, or 2 months before going live. We advise to keep track on the changes in your old identity provider. Before going live, you will then have to update only the changes, i.e. new users, password reset or remove unnecessary users.

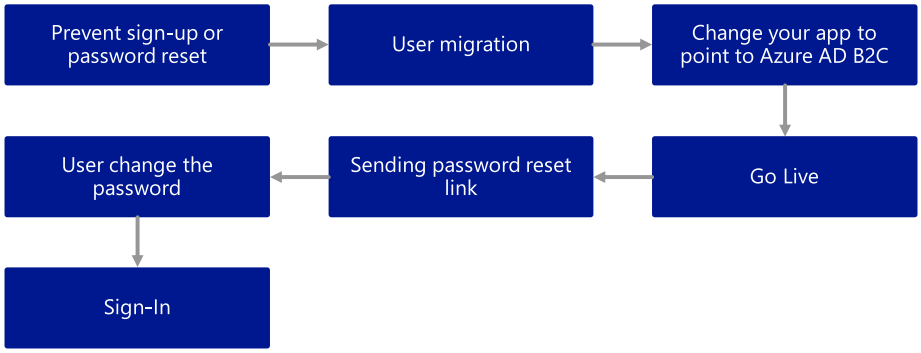
Migrating users without password and forcing them to change the password

Compared to the previous migration flows, this second flow fits when you don't have access to user's password. For example, the passwords are stored in hash format or in an identity provider you don't have access to - The identity provider allows to validate user credentials by calling an (out-of-band) web service -.

This flow also implies a pre-migration-process. It basically implies:

- First pre-migrating the users with the same way when you have the passwords (see previous section): you read the user data, including the user name and the user profile and create on that basis a new account in your B2C tenant with a random password you generate, prior to migrate to Azure AD B2C.
- Changing your mobile or web application(s) to authenticate against your B2C tenant by executing a sign-in (custom) policy.

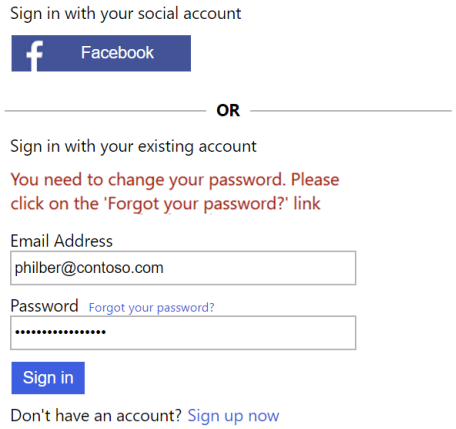
Once the pre-migration process is completed, you direct the users to a password reset policy to change their password.



Before going live, you should also prevent users from updating their password or creating new ones. The same considerations apply.

To force the users to change their password, the process involves custom policies with two calls to a RESTful API service. You indeed need to:

- Optionally send them a link to reset password policy.
- On the sign-in policy flow, add a call to a custom RESTful API. The API checks if the user migrates with random password – the user still didn't change the password -. If the user needs to change the password, the API will return a friendly error message to the end-user.



- On password reset policy flow, make an additional call to RESTful API to change the user status.

For that purpose, the above custom RESTful API is declared in your custom policy, e.g. the B2C_1A_TrustFrameworkExtensions custom policy, as a claims provider with two technical profiles - A technical profile can be indeed seen as a function -: one for the call during the sign-in flow and another one for the call during the password reset flow.

These technical profiles, respectively named in our illustration "LocalAccountSignIn" and "LocalAccountPasswordReset", can be used as a validation technical profile for the technical profiles used in the related flows as follows.

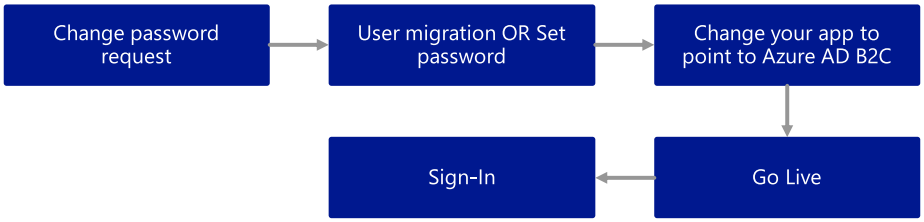
```
<TechnicalProfile Id="SelfAsserted-LocalAccountSignin-Email" >
   <ValidationTechnicalProfiles>
      <ValidationTechnicalProfile ReferenceId="LocalAccountSignIn" />
   </ValidationTechnicalProfile>
</TechnicalProfile>

<TechnicalProfile Id="LocalAccountWritePasswordUsingObjectId" >
   <ValidationTechnicalProfiles>
      <ValidationTechnicalProfile ReferenceId="LocalAccountPasswordReset" />
   </ValidationTechnicalProfile>
</TechnicalProfile>
```

Migrating users with password harvesting

With this third migration flow, you force users to change on your old identity provider their password. You capture the password and create a new account with the new password (or update the new password, if user already exists) in your B2C tenant.

The migration flow is only suitable for active users. If a user didn't change the password during that period of time, the account will not migrate to your B2C tenant.

One should note that you can combine and can run the second migration as well to cover 100% of the users.



Pre-migrating users without password and setting password just-in-time

Just-in-time (JIT) migration comes to deal with the case you don't have access to users' password in more efficient way. The process also involves a custom policy with a call to RESTful API service.

This migration flow enables to:

- Migrate user without password, set password just-in-time.

-or-

- Do a just-in-time migration.

The next sections examine in order these two options.

Migrating user without password, setting password just-in-time

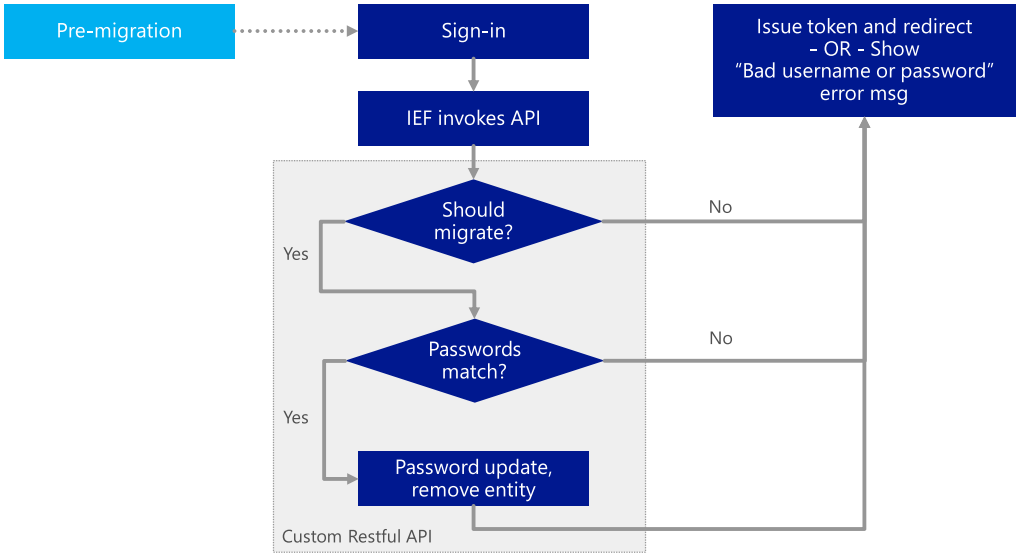
With this first option, you first you run the pre-migration, i.e. read the users from your old identity provider, create new accounts in your B2C tenant. But, since you do not have the passwords, you create the account with random password you generate. User's password will be set later just-in-time when users try to sign in.

For that purpose, you configure your custom policy to call a RESTful API endpoint sending the sign-in name and the password as input parameters on the function call. This API offers a simple business logic as follows.

On sign-in, when a user provides the username and password, your endpoint receives the user name and password. Then, it first checks if the user exists in the old identity provider. If yes, it validates the received credentials against the old identity provider. The validation is done by comparing the password in hash format or calling an out-of-band web service to run on the old identity provider.

If there is a match, i.e. the password provided by the user is valid, it then updates the password "just-in-time". Otherwise, you do nothing: you just let the Identity Experience Framework in Azure AD B2C to throw a "Bad user name or password" error message.

Et voilà! After the password is updated, the control returns back to Identity Experience Framework in Azure AD B2C for the rest of policy execution, which validates the credentials and issues an access token.



To avoid repeating the same process over and over again, right after the just-in-time migration, you need to change a user status (to migrated) or remove the entity from your old identity provider.

As before, you should prevent users from creating new accounts before going live.

Doing a just-in-time migration

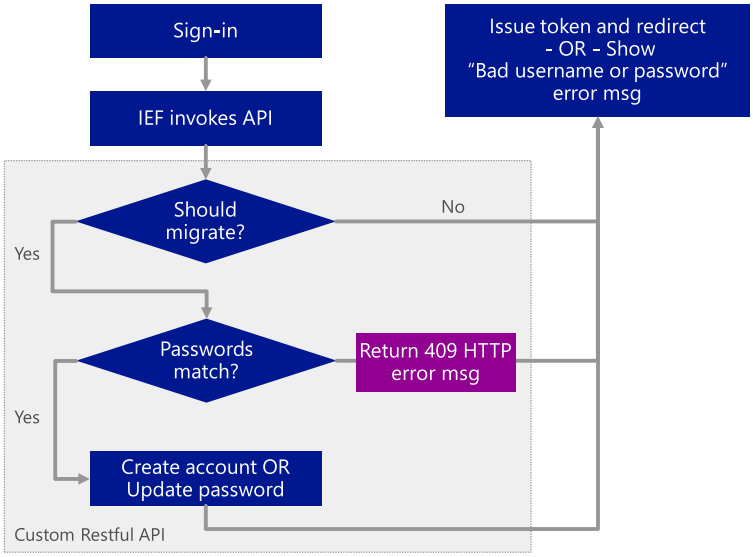
With this second option, you support a full just-in-time migration: not only the password is reset just-in-time, but this user account is also created just-in-time. The migration flow is only suitable for active users.

The related overall process remains the same as the previous one, but this time you don't run the pre-migration. Rather on sign-in, the RESTful API endpoint both creates the user and set the password.

The sign-in custom policy is almost the same as the one for setting the password just-in-time. If the password matches, you create the user with the password provided by the user.

However, if the password doesn't match, you return a "bad user name or password" 409 HTTP error message to the Identity Experience Framework in Azure AD B2C.

On the next try, i.e. when the user gets the bad username or password error message, you create the user or set the password. Unless the Identity Experience Engine will tell the user, the account you provided doesn't exist. So, you are responsible to return the error message.



You also need to create the user on the sign-up and password reset user journeys. So adequate custom policies need also to be configured to cover these scenarios.

On the sign-up custom policy, the user should be prevented from creating an existing account. A call is made to the RESTful API endpoint to check if the account exists in your old identity provider. If yes, a 409 HTTP error message "The account you provided already exist" is returned to the Identity Experience Engine in Azure AD B2C.

On the password reset, the account should already exist in your B2C tenant. Azure AD B2C will not allow to change the password for a non-existing user. This said, since the user may exit or abandon the password reset process, you should keep track on the users with random password and on the sign-in process, your code may need to update the password, instead of creating a new account. This is a (very) rare case but you should cover all the scenarios…

Considering all the above, our recommendation is to use the first approach, by combining the pre-migration with set password just-in-time.

Choosing the right path

The following table sums up the above migration flows:

Migration Flow	Pre-migration	Custom policy	Password accessible	Active users only	Complexity	User impact	Downtime	Retired old IdP
Migrate users with existing password	Yes		Yes		Low		Low	Yes
Migrate user without password, force them to change password	Yes				Low	Yes	Low	Yes
Migrate user with password harvesting	Yes			Yes	Low			
Pre-Migrate user without password, set password just-in-time	Yes	Yes			Medium		Low	
Do a just-in-time migration		Yes		Yes	High			

Based on this "background", let's now illustrates some of them with the tooling/code sample that comes with the "Starter Pack" along with the use of the Azure AD Graph API as discussed in the third document of this series.

Migrating users identified using a local IdP to your B2C tenant

This section is intended as an illustration of the first migration flow discussed earlier (see section § *Migrating users with existing password* earlier in this document).

For that purpose, we use as detailed hereafter an intermediary file formatted in JSON that contains all the key information about the users to migrate. In a real-life scenario, we will rather use a direct communication between the application used for the migration and the old identity provider (database).

This procedure follows a two-stage approach:

1. Retrieve the key information about users (sign in name, email, password, display name, given name, and surname) from the local identity provider/directory and save this information as a JSON array in a text file.

   You should not include domain-specific data such as object IDs, although you can include the attributes that your organization utilizes, such as job title, telephone number, and office location
   You can perform this task by capturing the JSON formatted output of PowerShell commands, (as shown in the section § *Using the Azure AD Graph API from PowerShell* in the third document of this series) and editing the results, or by following the principles also used in the third document by the

B2CGraphClient code sample application and writing the data to a file rather than displaying it on the screen. If necessary, an administrator can modify the resulting file and remove any records for users that should not be migrated.

2. Iterate through the records in the JSON text file and use the Azure AD Graph API to create the corresponding users in Azure AD. Again, you can adapt the techniques used by the B2CGraphClient application to achieve this.

The steps that follow assume that you have already retrieved the users to be migrated and saved them in a .json file with known passwords.

Following is an illustration of such a .json file. We will refer to this file as the *UserData.json* file.

```
{
  "Users": [
    {
      "email": "James@contoso.com",
      "displayName": "James Davis",
      "firstName": "James",
      "lastName": "Davis",
      "password": "1234567"
    },
    {
      "email": "Linda@contoso.com",
      "displayName": "Linda Taylor",
      "firstName": "Linda",
      "lastName": "Taylor",
      "password": "1234567"
    },
    {
      "email": "William@contoso.com",
      "displayName": "William Martin",
      "firstName": "William",
      "lastName": "Martin",
      "password": "1234567"
    },
    {
      "email": "Thomas@contoso.com",
      "displayName": "Thomas Lee",
      "firstName": "Thomas",
      "lastName": "Lee",
      "password": "1234567"
    },
    {
      "email": "Lisa@contoso.com",
      "displayName": "Lisa Bell",
      "firstName": "Lisa",
      "lastName": "Bell",
      "password": "1234567"
    },
    {
      "email": "Emily@contoso.com",
      "displayName": "Emily King",
      "firstName": "Emily",
      "lastName": "King",
      "password": "1234567"
    }
  ]
}
```

Notice that it includes users' "passwords".

Building the AADB2C.UserMigration code sample application

For the sake of this illustration, you will use a sample application that reads a .json file and uploads the users found to your B2C tenant, i.e. the AADB2C.UserMigration sample application that is provided as the *AADB2C.UserMigration.zip* archive file as part of the "Starter Pack" under the *\scenarios\aadb2c-user-migration* folder.

This application provides options to enable you to reuse the existing passwords for users (if they are available). The application requires an application registration in Azure AD B2C.

To save time, these steps reuse the B2CGraphClient created build and configured in the third document of this series.

Perform all the instructions that pertains to the section *§* Building and running the AADB2C.UserMigration project in the **Error! Reference source not found.**.

Using the AADB2C.UserMigration code sample application

> **Note** The following steps are primarily concerned with handling users with passwords that can be migrated. The cases for users with indecipherable passwords, or that require users to reset their password, are covered in the section § *Requiring users to change password on first sign-in* later in this document.

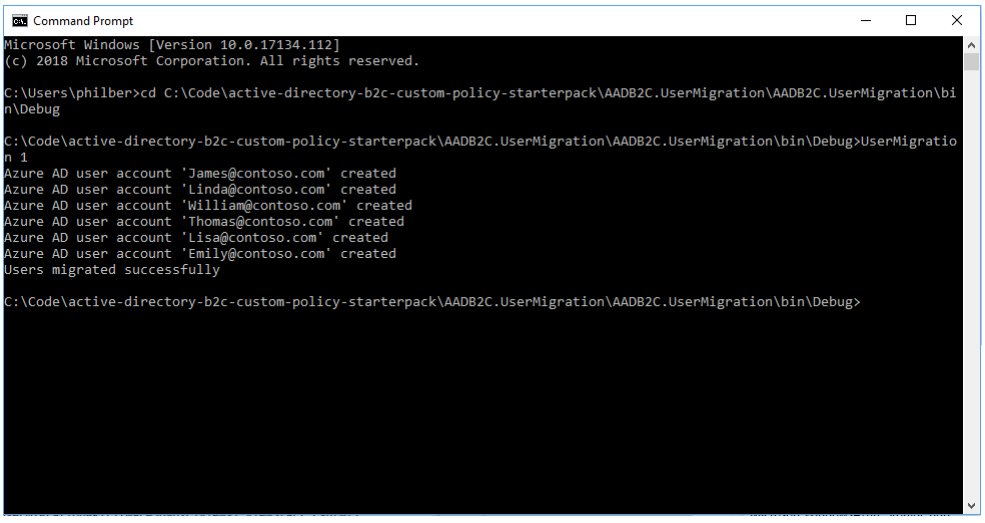
The rest of this section assumes that you successfully performed all the task outlined in section *§* Building and running the AADB2C.UserMigration project in the **Error! Reference source not found.**.

Proceed with the following steps:

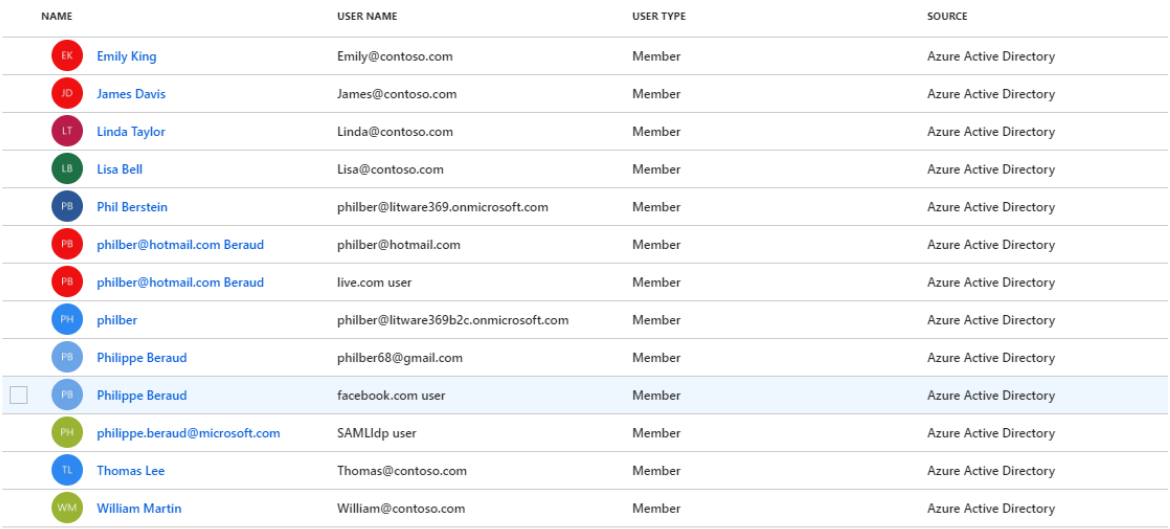
1. Using Windows Explorer, copy the above *UsersData.json* file containing the sample user data to the *AADB2C.UserMigration\bin\Debug* folder.
2. Open a command prompt window and move to the *AADB2C.UserMigration\bin\Debug* folder.
3. At the command prompt, type the following command.

```
C:\> UserMigration 1
```

   This command reads the records from the file *UsersData.json* and creates the corresponding users in your B2C tenant. You should see messages confirming each user as they are created.



4. In the Azure portal, make sure you are connected as a user account with administrative privileges in the B2C tenant. Select **Azure AD B2C**, and then select **Users** under **MANAGE**.
5. Verify that the new users appear in the list of users for your B2C tenant.



NAME	USER NAME	USER TYPE	SOURCE
Emily King	Emily@contoso.com	Member	Azure Active Directory
James Davis	James@contoso.com	Member	Azure Active Directory
Linda Taylor	Linda@contoso.com	Member	Azure Active Directory
Lisa Bell	Lisa@contoso.com	Member	Azure Active Directory
Phil Berstein	philber@litware369.onmicrosoft.com	Member	Azure Active Directory
philber@hotmail.com Beraud	philber@hotmail.com	Member	Azure Active Directory
philber@hotmail.com Beraud	live.com user	Member	Azure Active Directory
philber	philber@litware369b2c.onmicrosoft.com	Member	Azure Active Directory
Philippe Beraud	philber68@gmail.com	Member	Azure Active Directory
Philippe Beraud	facebook.com user	Member	Azure Active Directory
philippe.beraud@microsoft.com	SAMLIdp user	Member	Azure Active Directory
Thomas Lee	Thomas@contoso.com	Member	Azure Active Directory
William Martin	William@contoso.com	Member	Azure Active Directory

Migrating users identified using a social networking account to your B2C tenant

If users are identified by using social networking accounts, the social networking Identity provider takes responsibility for authenticating them; your B2C tenant does not contain any password data for these accounts.

To migrate users with these types of account, you must recreate the data required by the social network identity provider to identify these users.

This information is usually held in the **userIdentites** property of each user in Azure AD B2C and is typically a combination of the name of the issuer (such as Facebook) and an issuer user id (an encoded, unique value that identifies the user to the social network identity provider). For example:

```
"userIdentities": [
  {
     "issuer": "Facebook.com",
     "issuerUserId": "MTIzNDU2Nzg5MA=="
  }
]
```

For the sake of this illustration, we will use the following sample user data.

```
{
  "Users": [
    {
      "accountEnabled": true,
      "displayName": "AAA BBB",
      "userPrincipalName" : "aabb@litware369b2c.onmicrosoft.com",
      "passwordProfile": {
        "password": "Test1234",
        "forceChangePasswordNextLogin": false
      },
      "mailNickname" : "aabb",
      "userIdentities": [
        {
          "issuer": "facebook.com",
          "issuerUserId": "MATxTNg5MzYyMzMyMNY1Njc="
        }
```

```
      ]
    },
    {
      "accountEnabled": true,
      "displayName": "CCC DDD",
      "userPrincipalName" : "ccdd@litware369b2c.onmicrosoft.com",
      "passwordProfile": {
        "password": "Test1234",
        "forceChangePasswordNextLogin": false
      },
      "mailNickname" : "ccdd",
      "userIdentities": [
        {
          "issuer": "facebook.com",
          "issuerUserId": "NATxTNg5MzYyMzMyMNY1Njc="
        }
      ]
    },
    {
      "accountEnabled": true,
      "displayName": "EEE FFF",
      "userPrincipalName" : "eeff@litware369b2c.onmicrosoft.com",
      "passwordProfile": {
        "password": "Test1234",
        "forceChangePasswordNextLogin": false
      },
      "mailNickname" : "eeff",
      "userIdentities": [
        {
          "issuer": "facebook.com",
          "issuerUserId": "OATxTNg5MzYyMzMyMNY1Njc="
        }
      ]
    }
  ]
}
```

This above snippet contains a list of users in JSON formatted. Such a list typically constitutes an extract from the old identity provider database to migrate from.) The way the above data is formatted if how it is returned by using the Azure AD Graph API.)

Notice that each user in this file has a **userIdentities** property that references Facebook as a social identity provider. Do not change any data, expected the name of your B2C tenant in each **userPrincipalName** suffix in lieu of litware369b2c.onmicrosoft.com.

This content will be referred as to the *UsersSocialData.json* file.

To migrate these types of users to your B2C tenant, proceed with the following steps:

> **Important note** The AADB2C.UserMigration application does not currently support users with social identities, so these steps focus on using PowerShell instead.

1. Using Windows Explorer, copy the above *UsersData.json* file containing the sample user data to the *AADB2C.UserMigration\bin\Debug* folder.
2. Open a PowerShell command prompt window and move to the *AADB2C.UserMigration\bin\Debug* folder.

> **Note** For more information, see blog post WORKING WITH AZURE ACTIVE DIRECTORY GRAPH API FROM POWERSHELL[51].

3. Now open a PowerShell command prompt or a PowerShell ISE environment.

[51] WORKING WITH AZURE ACTIVE DIRECTORY GRAPH API FROM POWERSHELL:
https://blogs.technet.microsoft.com/paulomarques/2016/03/21/working-with-azure-active-directory-graph-api-from-powershell/

4. Create the following PowerShell function:

```
function GetAuthToken
{   param
    (
          [Parameter(Mandatory=$true)]
          $TenantName
    )
    $adal =
"${env:ProgramFiles}\WindowsPowerShell\Modules\ADAL.PowerShell\3.19.4.1\Microsoft.IdentityModel.Clients.ActiveDirectory.d
ll"
    [System.Reflection.Assembly]::LoadFrom($adal) | Out-Null
    $clientId = "1950a258-227b-4e31-a9cf-717495945fc2"
    $redirectUri = "urn:ietf:wg:oauth:2.0:oob"
    $resourceAppIdURI = "https://graph.windows.net"
    $authority = "https://login.windows.net/$TenantName"
    $PlatformParameters = New-Object Microsoft.IdentityModel.Clients.ActiveDirectory.PlatformParameters(0)
    $tokenCache = New-Object Microsoft.IdentityModel.Clients.ActiveDirectory.TokenCache
    $authContext = New-Object "Microsoft.IdentityModel.Clients.ActiveDirectory.AuthenticationContext" -ArgumentList
$authority, $tokenCache
    $authResult = $authContext.AcquireTokenAsync($resourceAppIdURI, $clientId, $redirectUri,
$PlatformParameters).GetAwaiter().GetResult()
    return $authResult
}
```

The **GetAuthToken** function prompts you to log in to the Azure AD (B2C) tenant specified by the **$TenantName** parameter and returns the authentication token if the login is successful.

5. Create a variable that references the name of your Azure AD tenant. Replace <your_B2C_tenant> with the name of your Azure AD tenant):

```
$tenant = "<your_B2C_tenant>.onmicrosoft.com"
```

6. Run the **GetAuthToken** function and extract the authentication token from the result:

```
$token = GetAuthToken -TenantName $tenant
```

7. Construct an HTTP authorization header object that contains the bearer token in the **$token** variable:

```
$authHeader = @{
   'Content-Type'='application\json'
   'Authorization'=$token.CreateAuthorizationHeader() }
```

8. Type the following commands:

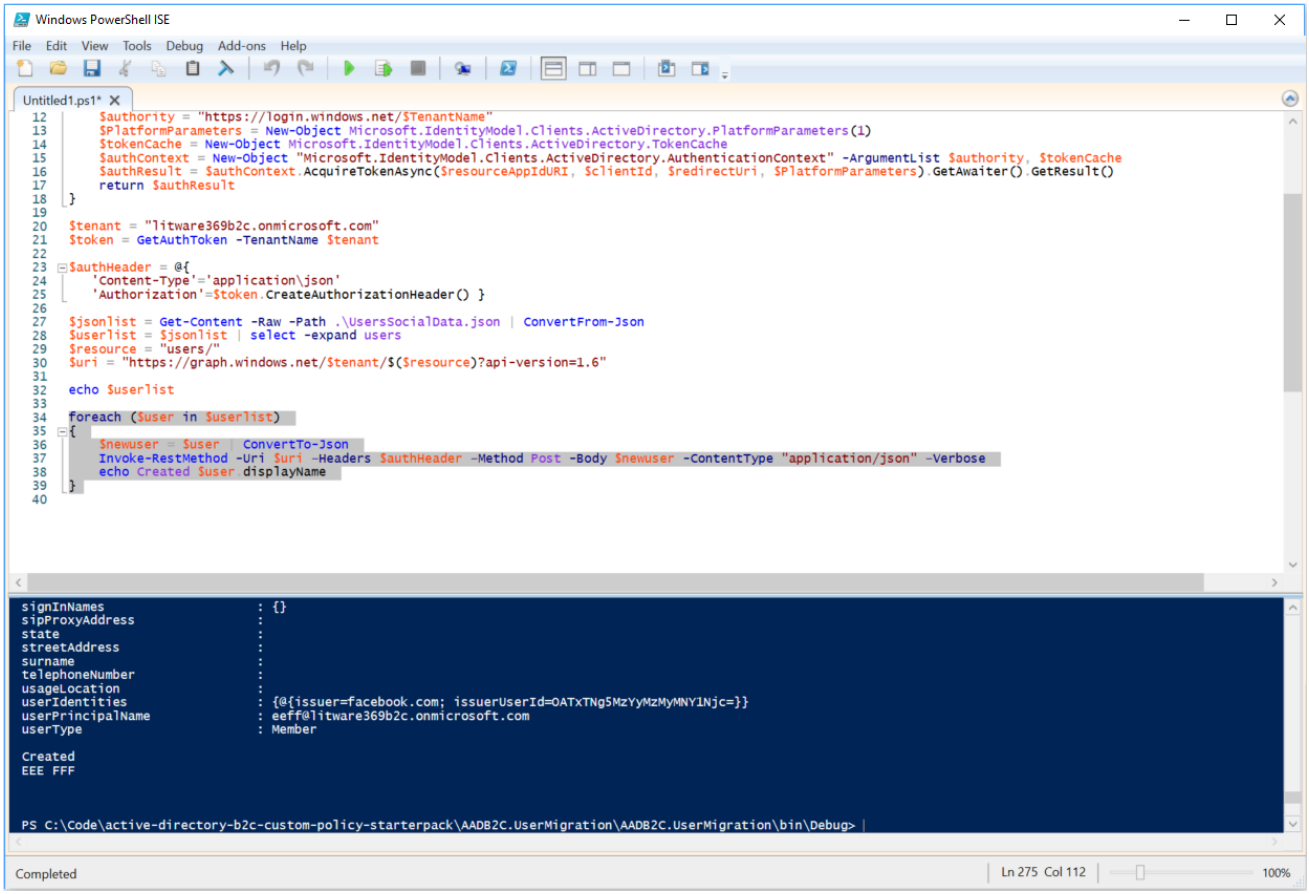
```
$jsonlist = Get-Content -Raw -Path .\UsersSocialData.json | ConvertFrom-Json
$userlist = $jsonlist | select -expand users
$resource = "users/"
$uri = "https://graph.windows.net/$tenant/$($resource)?api-version=1.6"
```

These statements read the user accounts from the above .json file and generate a list of **user** objects. The **$resource** and **$uri** variables reference the URI in your tenant where the new users should be stored.

9. Enter the following block of code:

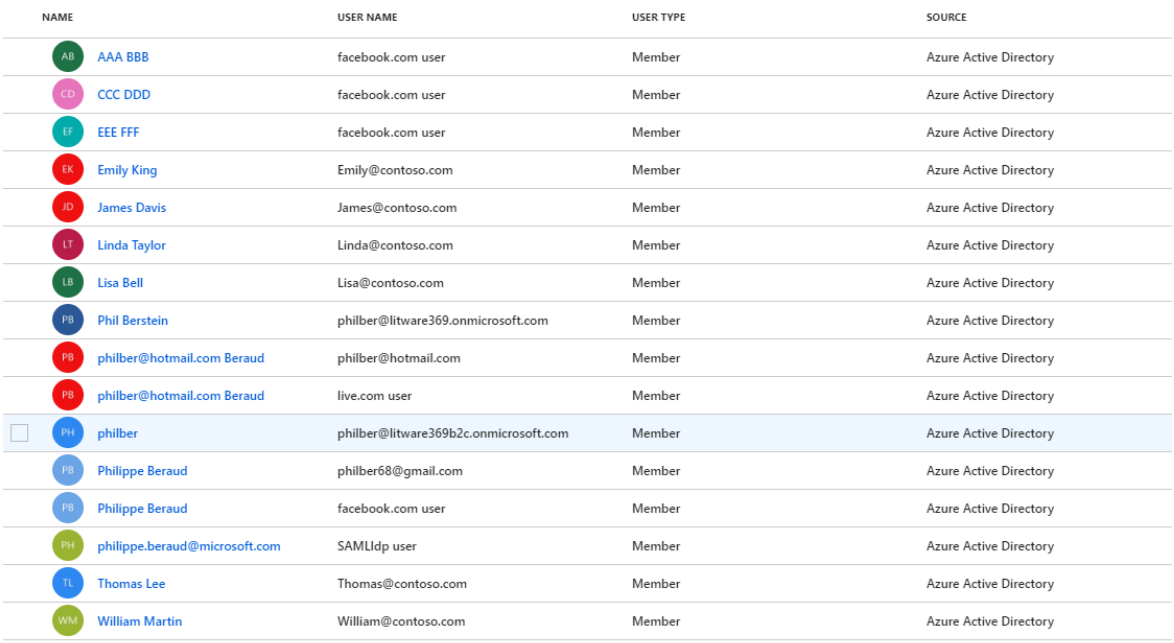
```
foreach ($user in $userlist)
{
    $newuser = $user | ConvertTo-Json
    Invoke-RestMethod -Uri $uri –Headers $authHeader –Method Post -Body $newuser -ContentType "application/json" –Verbose
    echo Created $user.displayName
}
```

These statements iterate through the list of user objects and send an HTTP POST request to your Azure AD to create each user in turn.



10. In the Azure portal, make sure you are connected as a user account with administrative privileges in the B2C tenant. Select **Azure AD B2C**, and then select **Users** under **MANAGE**.

11. Verify that the new users appear in the list of users for your B2C tenant, and that each user is identified as a facebook.com user.



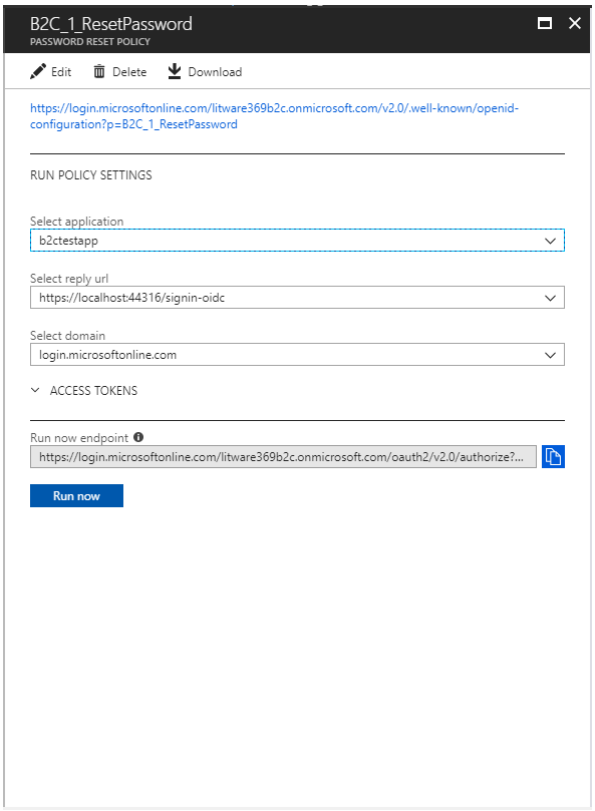
NAME	USER NAME	USER TYPE	SOURCE
AAA BBB	facebook.com user	Member	Azure Active Directory
CCC DDD	facebook.com user	Member	Azure Active Directory
EEE FFF	facebook.com user	Member	Azure Active Directory
Emily King	Emily@contoso.com	Member	Azure Active Directory
James Davis	James@contoso.com	Member	Azure Active Directory
Linda Taylor	Linda@contoso.com	Member	Azure Active Directory
Lisa Bell	Lisa@contoso.com	Member	Azure Active Directory
Phil Berstein	philber@litware369.onmicrosoft.com	Member	Azure Active Directory
philber@hotmail.com Beraud	philber@hotmail.com	Member	Azure Active Directory
philber@hotmail.com Beraud	live.com user	Member	Azure Active Directory
philber	philber@litware369b2c.onmicrosoft.com	Member	Azure Active Directory
Philippe Beraud	philber68@gmail.com	Member	Azure Active Directory
Philippe Beraud	facebook.com user	Member	Azure Active Directory
philippe.beraud@microsoft.com	SAMLIdp user	Member	Azure Active Directory
Thomas Lee	Thomas@contoso.com	Member	Azure Active Directory
William Martin	William@contoso.com	Member	Azure Active Directory

Requiring users to change password on first sign-in

This section is intended as an illustration of the second and forth migration flows discussed earlier (see sections § *Migrating users without password and forcing them to change the password* and § *Pre-migrating users without password and setting password just-in-time* earlier in this document).

As already outlined, to handle users with passwords that cannot be migrated, you will need to generate a random password, and then get users to reset their passwords when they log in. To achieve this, you have several strategies available, including:

- Directly emailing each user with the endpoint of the Password Reset built-in policy for your B2C tenant:



- Setting the **forceChangePasswordNextLogin** attribute of the password profile for the user to true:

```
{
   "accountEnabled" : true,
   "userPrincipalName" : "ddee@contoso.com",
   "displayName" : "DDD EEE",
   "passwordProfile" : {
       "password" : "********",
       "forceChangePasswordNextLogin" : true
   },
   "mailNickname" : "ddee"
}
```

In the situation where the passwords have been successfully migrated - the original passwords are reused - , it is still good practice to ask users to reset their passwords the next time they log in. You can use either of the strategies just highlighted. However, the problem with the first approach is that users might choose to ignore the email; there is no compulsion for them to reset their password.

The second approach enforces a password change, but from an administrative perspective it is not easy to determine that users have actually logged in and done so, so their accounts might still reference an old, unmodified password.

The AADB2C.UserMigration sample application introduced in section § *Migrating users identified using a local IdP to your B2C tenant* adopts a third approach that enables you to quickly check whether users have reset their passwords:

1. As users are migrated, a record of each user is also recorded in Azure table storage.
2. A separate REST API, AADB2C.UserMigration.API, implemented as part of the AADB2C.UserMigration solution reads the user records in table storage.
3. The next time the user logs in, a custom policy invokes the AADB2C.UserMigration.API with the details of the user.
    - If the user's record is found in table storage, the API returns the error message "You must change password".
    - Once the user has changed their password, the policy calls the API again to remove the user's record from table storage.

At any stage, an administrator with the appropriate access rights can read the data from table storage to determine which users are yet to reset their password.

Building the AADB2C.UserMigration code sample application

Perform all the instructions that pertains in the **Error! Reference source not found.** to the sections *§ Building and deploying the AADB2C.UserMigration.API project* and § *Updating the AADB2C.UserMigration project*.

Using the AADB2C.UserMigration code sample application

The rest of this section assumes that you successfully performed all the task outlined in section *§ Building and deploying the AADB2C.UserMigration.API project* in the **Error! Reference source not found.**.

For the sake of the illustration, the steps that follow assume that you have already retrieved the users to be migrated - and whose passwords are undecipherable and so need to have random passwords generated - and saved them to a .json file. We will refer to this file as the *UsersDataResetPasswords.json* file.

The sample user data are as follows:

```
{
 "Users": [
    {
      "email": "Fred@contoso.com",
      "displayName": "Fred Davis",
      "firstName": "Fred",
      "lastName": "Davis",
      "password": "12345678"
    },
    {
      "email": "Belinda@contoso.com",
      "displayName": "Belinda Taylor",
      "firstName": "Belinda",
      "lastName": "Taylor",
      "password": "12345678"
    },
    {
      "email": "Herbert@contoso.com",
      "displayName": "Herbert Martin",
      "firstName": "Herbert",
      "lastName": "Martin",
      "password": "12345678"
    },
    {
```

```
      "email": "Sid@contoso.com",
      "displayName": "Sid Lee",
      "firstName": "Sid",
      "lastName": "Lee",
      "password": "12345678"
    },
    {
      "email": "Sharon@contoso.com",
      "displayName": "Sharon Bell",
      "firstName": "Sharon",
      "lastName": "Bell",
      "password": "12345678"
    },
    {
      "email": "Emilia@contoso.com",
      "displayName": "Emilia King",
      "firstName": "Emilia",
      "lastName": "King",
      "password": "12345678"
    }
  ]
}
```

This file is similar in format to the *UsersData.json* file except that it contains a different set of user account information.

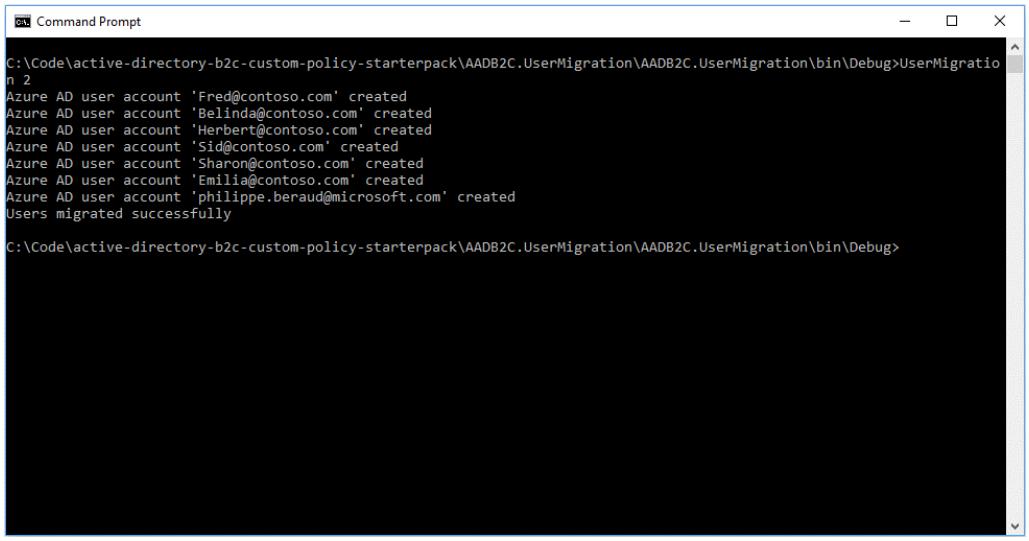
To illustrate this approach, proceed with the following steps:

1. Using Windows Explorer, copy the above *UsersDataResetPasswords.json* file containing the sample user data to the *AADB2C.UserMigration\bin\Debug* folder.
2. Open the *UsersDataResetPasswords.json* file using Visual Studio Code or Notepad and add a record for a user with your own email address. Save the file.
3. Open a command prompt window and move to the *AADB2C.UserMigration\bin\Debug* folder.
4. At the command prompt, type the following command.

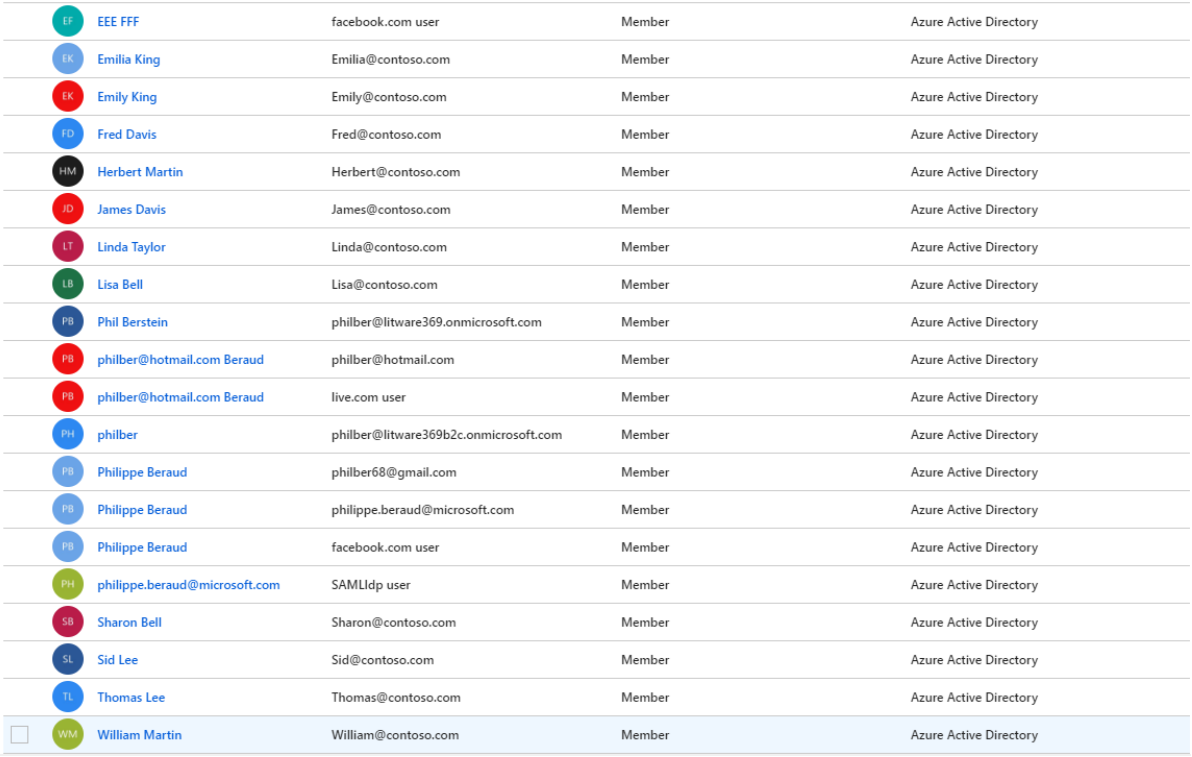
```
C:\> UserMigration 2
```

This command migrates the users listed in the *UsersDataResetPasswords.json file*. Remember that option 2 causes the program to run the **MigrateUsersWithRandomPasswordAsync** that you have just amended.

You should see messages confirming each user as they are created.

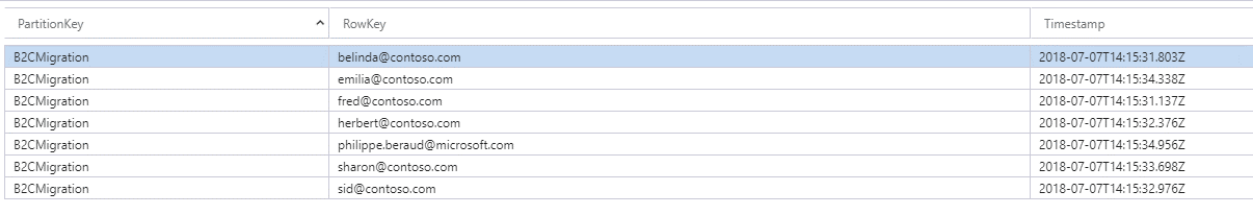


5. In the Azure portal, select **Azure AD B2C**, and then select **Users** under **MANAGE**.
6. Verify that the new users appear in the list of users for your B2C tenant.



EF	EEE FFF	facebook.com user	Member	Azure Active Directory
EK	Emilia King	Emilia@contoso.com	Member	Azure Active Directory
EK	Emily King	Emily@contoso.com	Member	Azure Active Directory
FD	Fred Davis	Fred@contoso.com	Member	Azure Active Directory
HM	Herbert Martin	Herbert@contoso.com	Member	Azure Active Directory
JD	James Davis	James@contoso.com	Member	Azure Active Directory
LT	Linda Taylor	Linda@contoso.com	Member	Azure Active Directory
LB	Lisa Bell	Lisa@contoso.com	Member	Azure Active Directory
PB	Phil Berstein	philber@litware369.onmicrosoft.com	Member	Azure Active Directory
PB	philber@hotmail.com Beraud	philber@hotmail.com	Member	Azure Active Directory
PB	philber@hotmail.com Beraud	live.com user	Member	Azure Active Directory
PH	philber	philber@litware369b2c.onmicrosoft.com	Member	Azure Active Directory
PB	Philippe Beraud	philber68@gmail.com	Member	Azure Active Directory
PB	Philippe Beraud	philippe.beraud@microsoft.com	Member	Azure Active Directory
PB	Philippe Beraud	facebook.com user	Member	Azure Active Directory
PH	philippe.beraud@microsoft.com	SAMLIdp user	Member	Azure Active Directory
SB	Sharon Bell	Sharon@contoso.com	Member	Azure Active Directory
SL	Sid Lee	Sid@contoso.com	Member	Azure Active Directory
TL	Thomas Lee	Thomas@contoso.com	Member	Azure Active Directory
WM	William Martin	William@contoso.com	Member	Azure Active Directory

7. Use the Microsoft Azure Storage Explorer[52] to verify that the new users also appear in the *aadb2cusermigration* Azure table.



PartitionKey	RowKey	Timestamp
B2CMigration	belinda@contoso.com	2018-07-07T14:15:31.803Z
B2CMigration	emilia@contoso.com	2018-07-07T14:15:34.338Z
B2CMigration	fred@contoso.com	2018-07-07T14:15:31.137Z
B2CMigration	herbert@contoso.com	2018-07-07T14:15:32.376Z
B2CMigration	philippe.beraud@microsoft.com	2018-07-07T14:15:34.956Z
B2CMigration	sharon@contoso.com	2018-07-07T14:15:33.698Z
B2CMigration	sid@contoso.com	2018-07-07T14:15:32.976Z

Updating the extension custom policy

Proceed with the following steps:

1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.

[52] Microsoft Azure Storage Explorer: https://azure.microsoft.com/en-us/features/storage-explorer/

2. Select the B2C_1A_TrustFrameworkExtensions custom policy file. A new blade opens.
3. Select **Download** and save the file as *TrustFrameworkExtensions.xml*.
4. Open the *TrustFrameworkExtensions.xml* file using an XML editor of your choice, for instance Visual Studio (Code).
5. Scroll down to the *ClaimsProviders* section.
6. Add the following <ClaimsProvider> element to the list in the *ClaimsProviders* section. Change <your_app> to ADB2CUserMigrationAPILitware369b2c (two occurrences) as per API publication, see section § *Building and deploying the AADB2C.UserMigration.API project* in the Appendix Building the code samples:

```
<ClaimsProvider>
   <DisplayName>Password Reset APIs</DisplayName>
   <TechnicalProfiles>
      <TechnicalProfile Id="LocalAccountSignIn">
        <DisplayName>Local account just in time migration</DisplayName>
          <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.RestfulProvider, Web.TPEngine, Version=1.0.0.0,
                                                Culture=neutral, PublicKeyToken=null" />
          <Metadata>

             <Item Key="ServiceUrl">https://<your_app>.azurewebsites.net/api/PrePasswordReset/LocalAccountSignIn</Item>

             <Item Key="AuthenticationType">None</Item>
             <Item Key="SendClaimsIn">Body</Item>
          </Metadata>
          <InputClaims>
             <InputClaim ClaimTypeReferenceId="signInName" PartnerClaimType="email" />
          </InputClaims>
          <UseTechnicalProfileForSessionManagement ReferenceId="SM-Noop" />
      </TechnicalProfile>

      <TechnicalProfile Id="LocalAccountPasswordReset">
         <DisplayName>Local account just in time migration</DisplayName>
         <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.RestfulProvider, Web.TPEngine, Version=1.0.0.0,
                                               Culture=neutral, PublicKeyToken=null" />
         <Metadata>

           <Item Key="ServiceUrl">https://<your_app>.azurewebsites.net/api/PrePasswordReset/PasswordUpdated</Item>

           <Item Key="AuthenticationType">None</Item>
            <Item Key="SendClaimsIn">Body</Item>
          </Metadata>
         <InputClaims>
            <InputClaim ClaimTypeReferenceId="email" PartnerClaimType="email" />
          </InputClaims>
          <UseTechnicalProfileForSessionManagement ReferenceId="SM-Noop" />
      </TechnicalProfile>
   </TechnicalProfiles>
</ClaimsProvider>
```

7. Add the following *ClaimsProvider* element to the list, directly after the one that you added in the previous step.

```
<ClaimsProvider>
   <DisplayName>Local Account</DisplayName>
   <TechnicalProfiles>
      <!-- This technical profile uses a validation technical profile to authenticate the user. -->
      <TechnicalProfile Id="SelfAsserted-LocalAccountSignin-Email">
         <ValidationTechnicalProfiles>
            <ValidationTechnicalProfile ReferenceId="LocalAccountSignIn" />
         </ValidationTechnicalProfiles>
      </TechnicalProfile>
      <TechnicalProfile Id="LocalAccountWritePasswordUsingObjectId">
         <ValidationTechnicalProfiles>
            <ValidationTechnicalProfile ReferenceId="LocalAccountPasswordReset" />
         </ValidationTechnicalProfiles>
      </TechnicalProfile>
   </TechnicalProfiles>
</ClaimsProvider>
```

8. Save the XML file.

Uploading the extension custom policy to your B2C tenant

Proceed with the following steps:

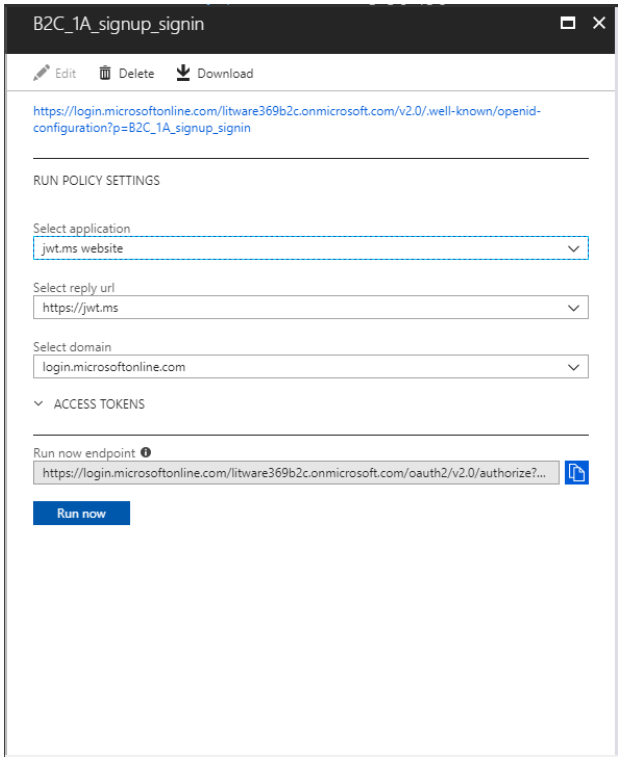
1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
2. Select **Upload Policy**.
3. Check **Overwrite the policy if it exists**.
4. In **Upload policy**, select the above *litware369b2c.onmicrosoft.com-B2C_1A_TrustFrameworkExtensions.xml* policy file.
5. Click **Upload** and ensure that it does not fail the validation.

Testing the custom policy by using Run Now

Proceed with the following steps:

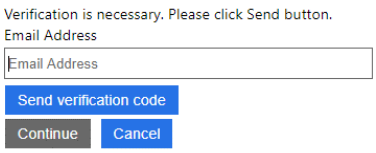
1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.

2. Click the **B2C_1A_signup_signin** custom policy to see its detail view. A new blade opens.

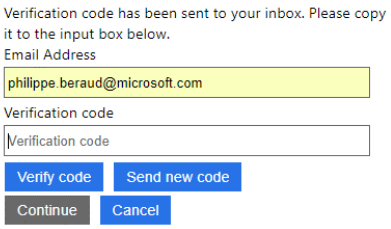


3. Click **Run now**.
4. Attempt to log in using your email address, as listed in the *UsersDataResetPasswords.json* file. Provide the password specified in this file, and then select **Sign in**.

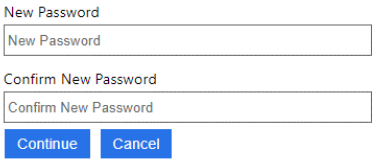
   You should see the error message "You need to change your password. Please click on the 'Forgot your password?' link below."
5. Close the sign-in page and return to the Azure portal.
6. Select the **B2C_1A_PasswordReset** custom policy, and then select **Run now**. Follow the process to reset the password for your email address.



   a. Specify your email address, as listed in the *UsersDataResetPasswords.json* file and click **Send verification code**.



b. Once you receive the verification code, enter it and then click **Verify code**. You can now change your password.



c. Now change your password and click **Continue**.

Decoded Token | Claims


```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid": "1dZZY23T1zabgVVbFSXB__xMlzUCG3XsYV5yIjeJmmU"
}.{
  "exp": 1530979695,
  "nbf": 1530976095,
  "ver": "1.0",
  "iss": "https://login.microsoftonline.com/cbf80679-8d13-4ac7-a5de-a4c1d69bd5d2/v2.0/",
  "sub": "ec25a319-3e33-4fdb-8629-7bc663208f60",
  "aud": "95b7953b-a499-48b9-bc1b-48bb095d6939",
  "acr": "b2c_1a_passwordreset",
  "nonce": "defaultNonce",
  "iat": 1530976095,
  "auth_time": 1530976095,
  "email": "philippe.beraud@microsoft.com"
}.[Signature]
```


7. When you have reset your password, return to the Azure portal, select the **B2C_1A_signup_signin** custom policy again, and then select **Run now**.

   Et voilà! You should now be able to sign in.

8. Eventually, verify with the Microsoft Azure Storage Explorer that your email address is no longer listed in the Azure table.

PartitionKey	RowKey	Timestamp
B2CMigration	belinda@contoso.com	2018-07-07T14:15:31.803Z
B2CMigration	emilia@contoso.com	2018-07-07T14:15:34.338Z
B2CMigration	fred@contoso.com	2018-07-07T14:15:31.137Z
B2CMigration	herbert@contoso.com	2018-07-07T14:15:32.376Z
B2CMigration	sharon@contoso.com	2018-07-07T14:15:33.698Z
B2CMigration	sid@contoso.com	2018-07-07T14:15:32.976Z

Helping to handle GDPR requirements

Since May 25, 2018, the General Data Protection Regulation[53], more commonly referred to by its English acronym "GDPR" is applicable. The GDPR imposes new rules on companies, government agencies, non-profits, and other organizations that offer goods and services to people in the European Union (EU), or that collect and analyze data tied to EU residents.

In the age of digital transformation, data privacy and improved security have become major concerns. The GDPR is fundamentally concerned with the issue of protecting the privacy of individuals and enabling them to exercise their rights in this regard. To this end, the GDPR establishes a set of the most stringent global requirements imposed on organizations in terms of protection of privacy. These requirements govern how you must manage and protect the personal data of individuals in the EU while respecting their individual choices, no matter where the data are processed, stored, or sent.

Thus, Microsoft and its customers have now set out on the path to achieve the privacy objectives set by the GDPR. Microsoft believes that privacy is a fundamental right, and we believe that the GDPR represents an important advance in terms of privacy and protection of related rights. At the same time, we recognize that the GDPR will impose significant changes on organizations around the world.

> **Note** For more information, see whitepapers GDPR HOW-TO: GET ORGANIZED AND IMPLEMENT THE RIGHT PROCESSES[54] and GDPR: HOW TO BECOME AND REMAIN COMPLIANT[55].

Although the path to GDPR may be difficult, Microsoft is here to help. Information about Microsoft Services to support your GDPR accountability as well as an understanding of the technical and organizational measures Microsoft has taken to support the GDPR are provided on the Microsoft Service Trust portal[56].

In this context, this section provides specific information regarding Azure AD B2C regarding the following topics:

- Explicit data subject consents (opt-in).
- Data subject requests (DSRs).
- Data breach notification.

The next sections cover each topic in order.

Getting data subjects' consent

The GDPR mandates data controllers (you) to obtain an explicit consent from individuals (or, data subjects) prior collecting and analyzing personal data.

Personal data must be indeed processed in a transparent manner in the sense that the data subjects must be informed of the purpose of the processing – uses for anything other than for the specific processing

[53] REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL OF 27 APRIL 2016 ON THE PROTECTION OF NATURAL PERSONS WITH REGARD TO THE PROCESSING OF PERSONAL DATA AND ON THE FREE MOVEMENT OF SUCH DATA, AND REPEALING DIRECTIVE 95/46/EC (GENERAL DATA PROTECTION REGULATION): http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32016R0679

[54] GDPR HOW-TO: GET ORGANIZED AND IMPLEMENT THE RIGHT PROCESSES: https://aka.ms/GDPR-process-EN

[55] GDPR: HOW TO BECOME AND REMAIN COMPLIANT: https://aka.ms/gdpr-become-compliant-en

[56] GET STARTED: SUPPORT FOR GDPR ACCOUNTABILITY: https://servicetrust.microsoft.com/ViewPage/GDPRGetStarted

described are not permitted – and that only the data necessary for that purpose will be collected. The consent of the individuals must be clearly and explicitly requested and obtained prior to the collection of the data and can be withdrawn at any time. Special conditions also apply for minors who require parental authority.

> **Important note** The GDPR permits Member States in the European Union (EU) to set the age at which minors require parental authority for consent as low as 13 years. So this requirement may vary from one Member State to the next.

Capturing "terms of use" (ToU) agreements

Azure AD B2C allows you to capture "terms of use" (ToU) agreements by tracking when users consent. You can also prompt users if their agreement is out of date or if they haven't signed it previously.

> **Note** For more information, see section CAPTURE TERMS OF USE AGREEMENT[57] in article MANAGE USER ACCESS IN AZURE AD B2C.

To incorporate an orchestration step to a Sign-Up and Sign-In (SUSI) user journey that prompts the user to accept the organization's terms and conditions, you need to add a related claim to the sign-up page, and then validate the input using an external RESTful API as already illustrated in section § *Integrating with a RESTful API* earlier in document.

For the sake of simplicity, we illustrate hereafter only the first part.

Updating the custom policies

Proceed with the following steps:

1. Navigate to the *SocialAndLocalAccounts* folder in the "Starter Pack".
2. Open the base policy file, i.e. the *TrustFrameworkBase.xml* file, using an XML editor of your choice, for instance Visual Studio (Code).
3. Scroll down to the *ClaimsSchema* node.
4. Inside the *ClaimsSchema* node, insert the following claim type.

```
<ClaimType Id="TnCs">
   <DisplayName>Terms of Service Consent</DisplayName>
   <DataType>string</DataType>
   <UserHelpText>I agree to the Litware 369  terms of service.</UserHelpText>
   <UserInputType>CheckboxMultiSelect</UserInputType>
   <Restriction>
      <Enumeration Text="I agree to the Litware 369  terms of service." Value="6/19/2018" SelectByDefault="false" />
   </Restriction>
 </ClaimType>
```

5. Find the element <TechnicalProfile Id="LocalAccountSignUpWithLogonEmail">, and in the *OutputClaims* node, add the following element.

```
<OutputClaim ClaimTypeReferenceId="TnCs" Required="true" />
```

[57] MANAGE USER ACCESS IN AZURE AD B2C: https://docs.microsoft.com/en-us/azure/active-directory-b2c/manage-user-accesst

6. Save the XML file.
7. To add a claim to the sign-up and sign-in (SUSI) page, now open the *SignUpOrSignIn.xml* relying party file using the XML editor of your choice.
8. Find the element <TechnicalProfile Id="PolicyProfile">, and in the *OutputClaims* node, add the following XML snippet.

```
<OutputClaim ClaimTypeReferenceId="TnCs"/>
```

9. Save the XML file.

Uploading the custom policies to your B2C tenant

Proceed with the following steps:

6. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.
7. Select **Upload Policy**.
8. Check **Overwrite the policy if it exists**.
9. In **Upload policy**, select the *TrustFrameworkBase.xml* policy file.
10. Click **Upload** and ensure that it does not fail the validation.
11. Repeat the above steps 2-5 with the *SignUpOrSignIn.xml* file.

Testing the custom policy by using Run Now

Proceed with the following steps:

1. In the Azure portal, in the **Azure AD B2C** blade, select **Identity Experience Framework - PREVIEW**.

2. Click the **B2C_1A_signup_signin** custom policy to see its detail view. A new blade opens.
3. Click **Run now**.
4. In the sign-in page, select **Sign-up now**.
5. In the sign-up page, notice that the **Terms of Service Consent** checkbox appears.
6. Attempt to create an account without checking **Terms of Service Consent**. This should fail with the message "This information is required." You should only be able to proceed once you have checked the box.
7. Verify that you can create an account if you select the **Terms of Service Consent** box.

Getting parental consent

As stated above, applications need consent to allow access for the minors. Minors must have consent from a parent to use any service, regardless of whether it's targeted at minors. In addition, telemetry, analytics and targeted marketing should be disabled for minors even after parental consent.

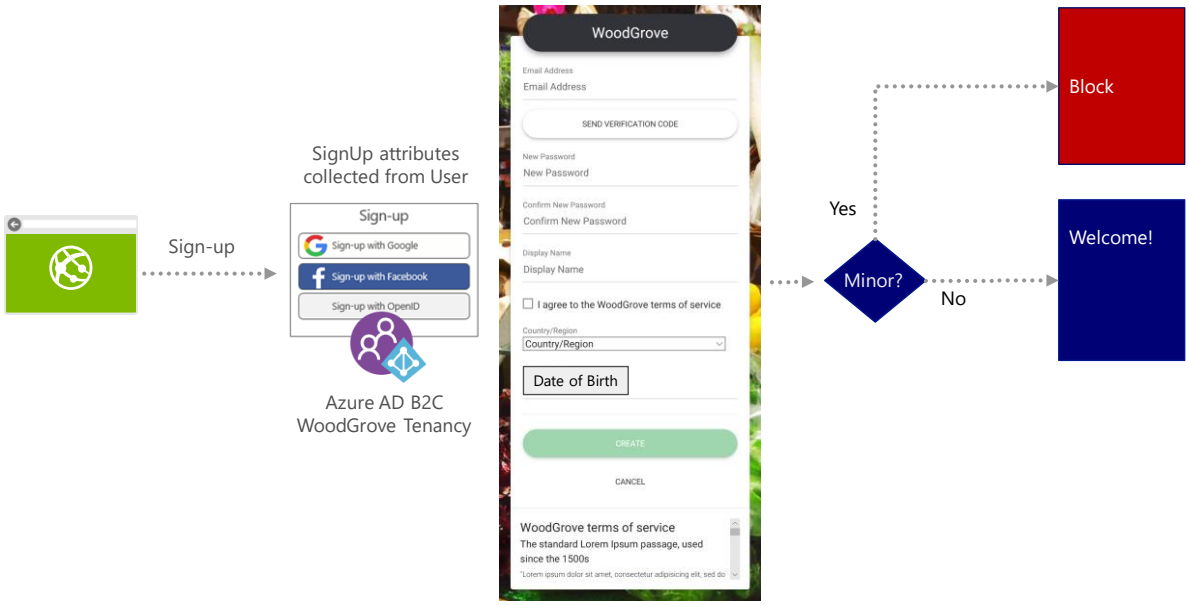
Azure AD B2C provides parental consent features as follows:

- Collecting Date of Birth and Country for all users to determine if the user is a minor. This information can be collected during sign-up for new users, or during sign-in for existing users if Date of Birth and Country have not been previously collected.

  This feature is illustrated in the Woodgrove Groceries demo[58] introduced at the end of the second document of this series.

- Blocking minors from signing up from your service if that is the desired behavior.

  This feature is also illustrated in the Woodgrove Groceries demo.



- Issuing id tokens with a claim indicating that the user is considered a minor. This will take into consideration that different countries have different laws about what age is considered a minor.
- Invoking a parental consent service over the OpenID Connect (OIDC) standard protocol. The age verification and parental consent itself is not a service provide by Microsoft.
- Storing a child's parent data in the directory through the Azure AD Graph API.
- Enabling and disabling a child's account through the Azure AD Graph API. This will be possible by flagging an account as **minor without parental consent.**
- Deleting a child's account.

[58] WoodGrove Groceries demo code: https://github.com/Azure-Samples/active-directory-external-identities-woodgrove-demo

Note For more information, see articles MANAGE USER ACCESS IN AZURE AD B2C[59], USING AGE GATING IN AZURE AD B2C[60], and MANAGE USER DATA IN AZURE AD B2C[61].

Getting consent to share data with 3rd party services

In addition to the above, the GDPR mandates consent to be given explicitly (opt-in) for sharing individual (data subjects) information with 3rd parties. Data subjects must be able to withdraw consent for sharing their personal data at any time.

As per above sections, custom policies allow for enforcing the collection of consent from a user. For example:

- Consent can be collected as part of all registrations (i.e. sign-up) and persisted in your B2C tenant.
- Consent can be collected if it is missing or has expired during a sign-in user journey.
- Access to the application can be blocked by the Identity Experience Framework in Azure AD B2C if consent is declined.

Fulfilling the Data Subject Requests (DSRs)

With the GDPR, the rights of the data subject are extended, starting with the consent that the person must provide in full knowledge of the facts: transparency is imposed on the purpose of the processing, the personal data collected, any data transfers to third parties and/or outside the European Union (EU), the retention period of personal data, and the right to submit a complaint.

The GDPR grants individuals (or, data subjects) certain rights in connection with the processing of their personal data, including the right to access their data, correct inaccurate data, erase data or restrict its processing, receive their data and fulfill a request to transmit their data to another controller. (In addition, the concept of profiling is introduced to indicate that the data subject must be informed of the said profiling and may refuse it except where this is necessary for the performance of the contract.)

The GDPR requires data controllers (you) and data processors (Microsoft in the context of Azure AD B2C) to respond to those requests. This section shows hereafter how Microsoft (for Azure AD B2C) will enable you to do so.

Note For more information, see article GDPR: DATA SUBJECT REQUESTS (DSRS)[62].

Additional support has been introduced in Azure AD B2C to the audit logs about user data both via APIs and through the Azure management portal. This, along with the changes made to how Microsoft handle data, helps you fulfill requests for deletion and accessing data.

[59] Ibid
[60] USING AGE GATING IN AZURE AD B2C: https://docs.microsoft.com/en-us/azure/active-directory-b2c/basic-age-gating
[61] MANAGE USER DATA IN AZURE AD B2C: https://docs.microsoft.com/en-us/azure/active-directory-b2c/manage-user-data
[62] GDPR: DATA SUBJECT REQUESTS (DSRS): https://servicetrust.microsoft.com/ViewPage/GDPRDSR

Note For more information, see articles Accessing Azure AD B2C audit logs[63] and Manage user data in Azure Active Directory B2C[64].

The features in Azure AD B2C do not guarantee GDPR compliance, rather, they must be used in specific ways to achieve compliance. Overall orchestration and reporting of GDPR compliance across all applications, databases, and customer support systems is not a goal.

The specific roles of Azure AD B2C will depend on the unique approach to compliance in each organization.

We illustrate hereafter how to fulfill the right to export and the one to be forgotten.

Fulfilling the right to export

Providing the ability to exercise the right to export with your B2C tenant requires to both:

- Get the current state of user with a read of the user information via the Azure AD Graph API as covered in the third document of this series.
- Get the user's activity from the improved audit logs.

Developers (i.e. app creators) need to call the Azure AD Graph API for the user. B2C users do not interact with the Azure AD Graph API directly.

The Azure AD Graph API may be used to export the user data in your B2C tenant in real time. Alternatively, an end user may log into to an Azure AD B2C registered application, proactively select a "Profile View" policy and review their own provided data as part of an user journey.

Furthermore, the usage logs (a.k.a. Audit log) may be downloaded by the application using the Azure AD Reporting API, filtered by the object ID of a user and provided to the end user in a compliant industry standard format.

Note For more information, see article Get started with the Azure Active Directory reporting API[65].

[63] Accessing Azure AD B2C audit logs: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-reference-audit-logs
[64] Manage user data in Azure Active Directory B2C: https://docs.microsoft.com/azure/active-directory-b2c/manage-user-data
[65] Get started with the Azure Active Directory reporting API: https://docs.microsoft.com/en-us/azure/active-directory/active-directory-reporting-api-getting-started-azure-portal

Following is an example of a B2C export script in PowerShell. This is an example only and will not work in its present form. It should be used to form your own query by updating the variables outlined in red:

```
# Constants

# Insert your application's Client ID, a GUID registered by Global Admin granted read permissions granted on Azure AD Graph
$ClientID       = "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
# Insert your application's Client Key/Secret string
$ClientSecret   = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
# Insert your Azure AD Tenant; for example, contoso.onmicrosoft.com
$tenantdomain   = "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
# Insert the ObjectID of the user to dump.
$userToDump     = "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"

# AAD Instance; for example https://login.microsoftonline.com
$loginURL       = "https://login.microsoftonline.com"
# Azure AD Graph API resource URI
$resource       = "https://graph.windows.net"

Write-Output "Dumping user $userToDump from Directory: $tenantdomain"
#
# Create HTTP header, get an OAuth2 access token based on client id, secret and tenant domain
#
$body       = @{grant_type="client_credentials";resource=$resource;client_id=$ClientID;client_secret=$ClientSecret}
$oauth      = Invoke-RestMethod -Method Post -Uri $loginURL/$tenantdomain/oauth2/token?api-version=1.0 -Body $body
$baseURL = 'https://graph.windows.net/'+$tenantdomain+'/'
#
# Parse audit report items, save output to file(s): userX.json, where X = 0 thru n for number of nextLink pages
#
if ($oauth.access_token -ne $null) {
    $i=0
    $headerParams = @{'Authorization'="$($oauth.token_type) $($oauth.access_token)"}
    #
    # This line dumps a single user
    #
    $url = 'https://graph.windows.net/' + $tenantdomain + '/users/'+$userToDump+'?api-version=1.6'
    #
    # this line will dump all the users in the tenant.
    #
    #$url = 'https://graph.windows.net/' + $tenantdomain + '/users/?api-version=1.6&$top=5'
    # loop through each query page (1 through n)
    Do{
        # display each event on the console window
        Write-Output "Fetching data using Uri: $url"
        $aUser = (Invoke-WebRequest -UseBasicParsing -Headers $headerParams -Uri $url)
        foreach ($userProperties in ($aUser.Content | ConvertFrom-Json)) {
            Write-Output ($userProperties | ConvertTo-Json)
        }

        # save the query page to an output file
        Write-Output "Save the output to a file user$i.json"
        $aUser.Content | Out-File -FilePath user$i.json -Force
        $url = ($aUser.Content | ConvertFrom-Json).'odata.nextLink'
        if ( $url -ne $null ){ $url = $baseURL + $url + '&api-version=1.6&$top=5'; Write-output "Next Page: $i $url"}
        $i = $i+1
    } while($url -ne $null)
} else {
    Write-Host "ERROR: No Access Token"
}
```

Fulfilling the right to be forgotten

Likewise, providing the ability to exercise the right to be forgotten with your B2C tenant simply requires deleting the user account via the Azure AD Graph API.

Similarly, developers (i.e. app creators) need to call Azure AD Graph API for the user. B2C users do not interact with Azure AD Graph API directly.

The DELETE command in the Azure AD Graph API enables you to delete all the user data for a user in response to a qualified request. The data will be soft-deleted immediately before acknowledging success,

and all data will be hard deleted within 30 days. You may also choose to hard delete the user data. In this case, the data will be deleted within one day. Soft-delete is reversible within 30 days, hard-delete is not.

Usage logs (a.k.a. audit logs) are deleted automatically every 30 days by the Azure AD B2C service for all users.

Handling breach notification

The GDPR imposes data controllers (you) to notify any data breach within 72 hours of having become aware of a breach.

In the event of a breach, Microsoft will contact the Azure subscription owner as well as the security contact specified. **This requires that the Azure AD B2C tenant be linked to an Azure subscription and that the information is up to date**.

> **Note** For more information, see article DATA BREACH NOTIFICATION UNDER THE GDPR[66].

This concludes this fourth document of this series.

[66] DATA BREACH NOTIFICATION UNDER THE GDPR: https://servicetrust.microsoft.com/ViewPage/GDPRBreach

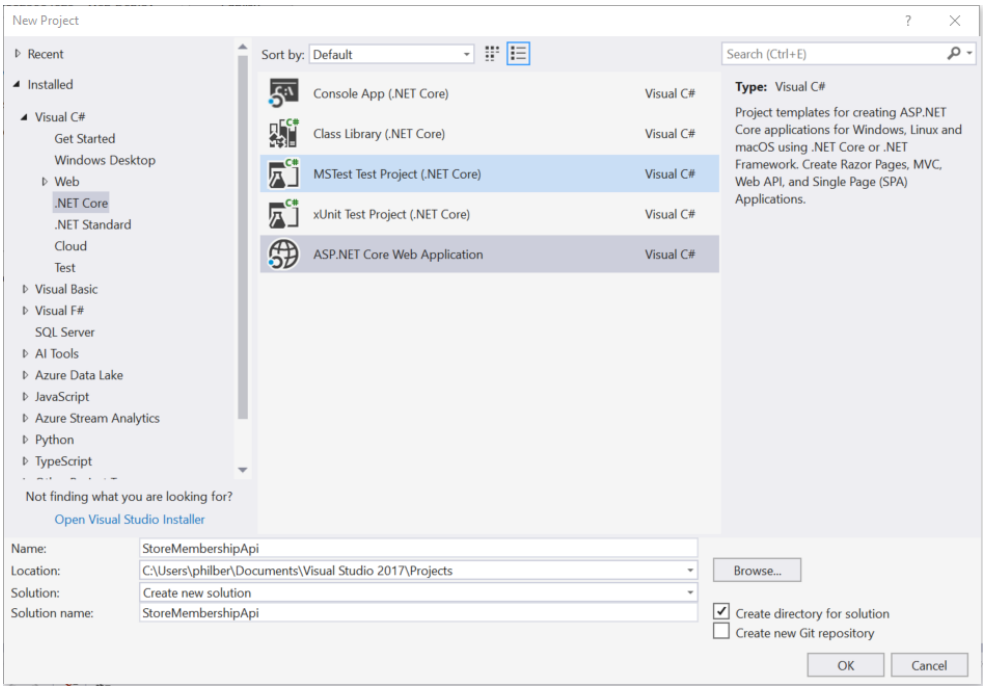
Appendix Building the code samples

Building a RESTful API claims provider

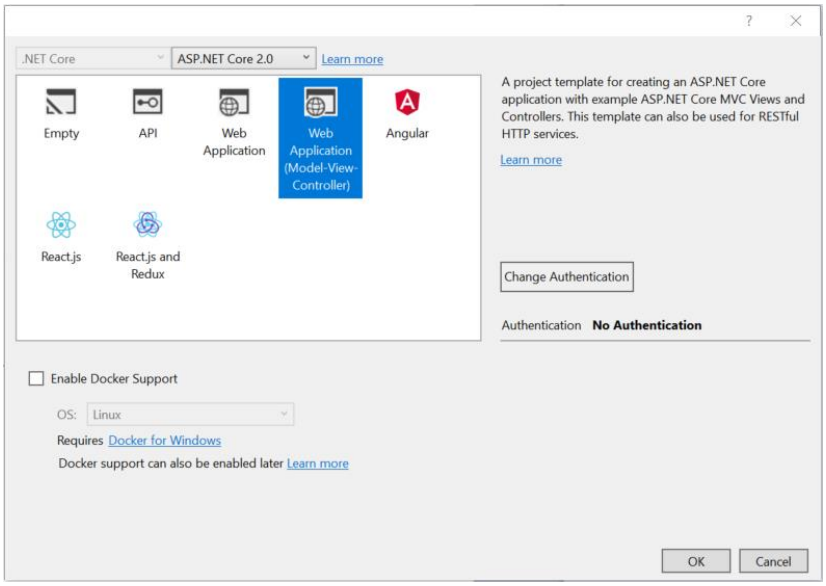
Creating a new API in ASP.NET Core

To create a new RESTful API in ASP.NET Core, proceed with the following steps:

1. Start Visual studio 2017.
2. On the **File** menu, select **New** and then select **Project**.
3. In the **New Project** dialog box, expand **Visual C#**, select **Web**, and then select **ASP.NET Core Web Application.**
4. Name the project **StoreMembershipApi**, and then select **OK**.



5. In the **New ASP.NET Core Web Application** dialog box, select **Web Application (Model View Controller)**, select **No Authentication**, and then select **OK**.



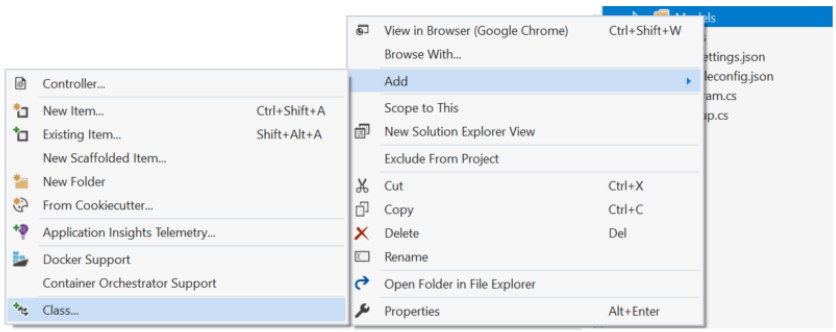
Adding data models and the controller to the API project

The user will provide a Store Membership Number at the time of registration of his account. The RESTful API will validate this number, and if it is verified, registration will be allowed to proceed. If the number is invalid, the service will send back a *validation failed* error.

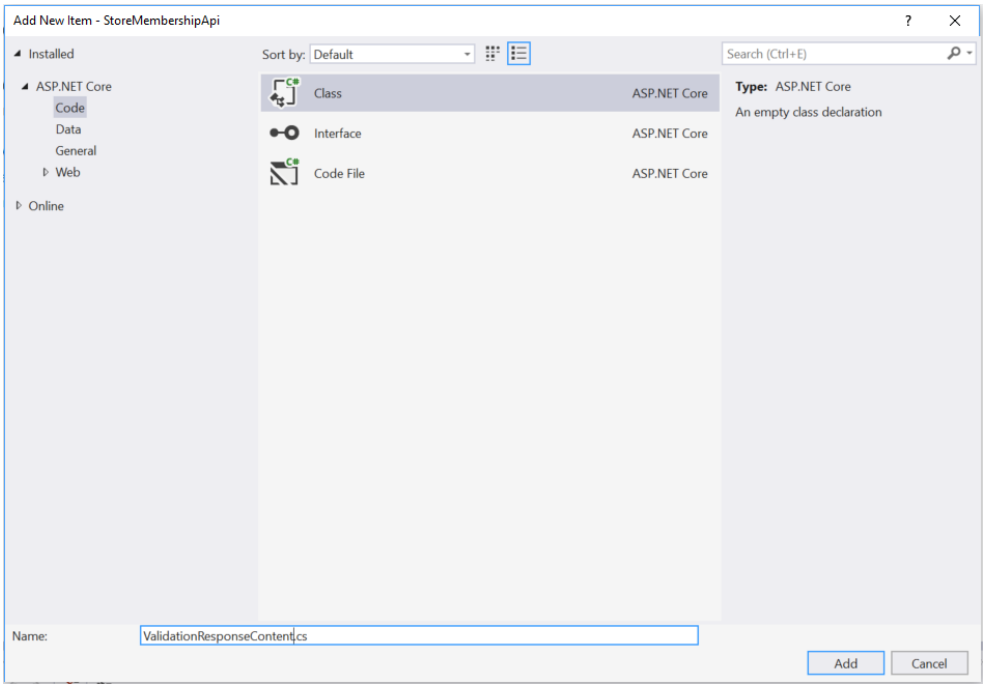
For the sake of this illustration, the validation is simple: the provided Store Membership Number must be an integer that is divisible by 5.

Proceed with the following steps:

1. In Solution Explorer, right-click **Models**, select **Add**, and the select **Class.**



2. In the **Add New** Item dialog box, select **Class**. Specify the name **ValidationResponseContent**, and then select **Add**.



3. Replace the code in the *ValidationResponseContent.cs* file with the following code snippet:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace StoreMembershipApi.Models {
   public class ValidationResponseContent : StoreResponseContent
   {
       public string StoreMembershipNumber { get; set; }
   }
}
```

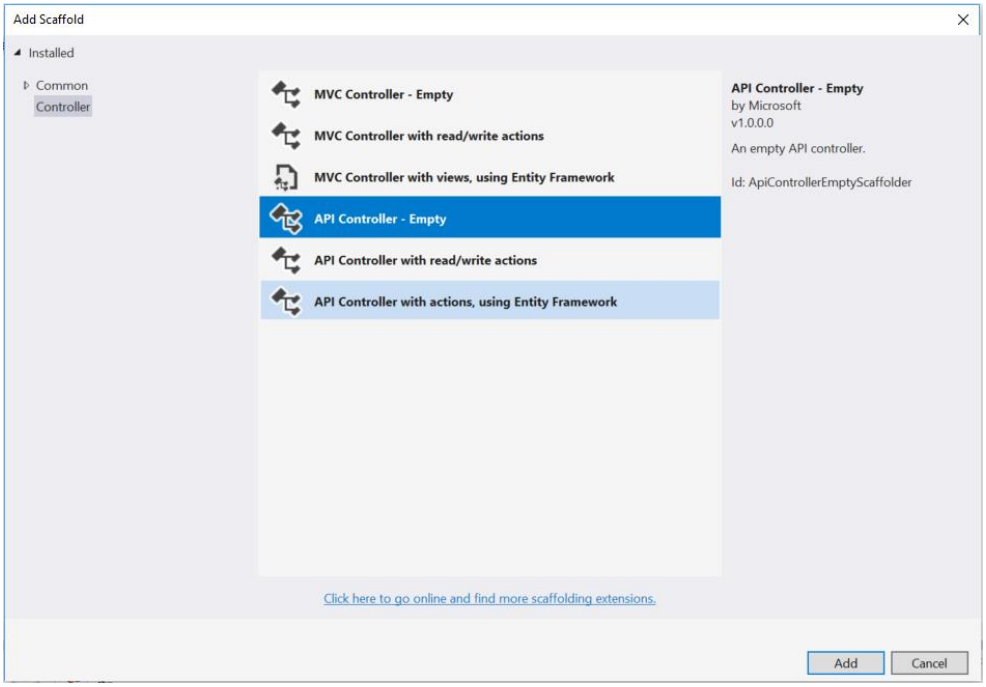
4. Using the same procedure, add another class to the **Models** folder, named **StoreResponseContent**. Replace the code for the class with the following code snippet:

```
using System.Net;
using System.Reflection;

namespace StoreMembershipApi.Models
{
   public class StoreResponseContent
   {
      public string Version { get; set; }
      public int Status { get; set; }
      public string UserMessage { get; set; }

      public StoreResponseContent()
      { }

      public StoreResponseContent(string message, HttpStatusCode status)
      {
         this.UserMessage = message;
         this.Status = (int)status;
         this.Version = Assembly.GetExecutingAssembly().GetName().Version.ToString();
      }
   }
}
```

5. Add a further class to the **Models** folder, named **MembershipRequest**. Replace the code generated for this class with the following code snippet:

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace StoreMembershipApi.Models
{
    public class MembershipRequest
    {
        public int StoreMembershipNumber { get; set; }
    }
}
```

6. In Solution Explorer, right-click the **Controllers** folder, select **Add**, and then select **Controller**.
7. In the **Add Scaffold** dialog box, select **API Controller -Empty**, and then select **Add**.



8. In the **Add Empty API Controller** dialog box, name the controller **MembershipController**, and then select **Add**.

9. Replace the code generated for the controller with the following code snippet:

```
using System; using System.Net;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using StoreMembershipApi.Models;

namespace StoreMembershipApi.Controllers
{
   /// <summary>
   /// This controller is responsible for responding to requests from our custom policies
   /// to validate store membership numbers and retrieve customer membership dates.
```

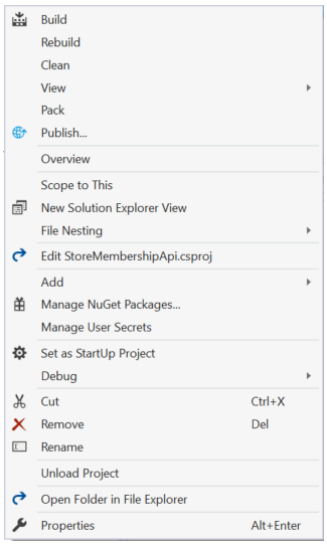
```
   /// </summary>

   [Route("api/[controller]")]
   public class MembershipController : Controller
   {
      /// <summary>
      /// This method receives a storeMembershipnumber from the policy validation step and returns a 409 Conflict
      /// response if the store membership number is not valid (not a multiple of 5), and a 200 Ok response on
      /// successful validation of the store membership number.
      /// </summary>
      /// <param name="request">Passed from the policy validation step, contains a storeMembershipNumber.</param>
      /// <returns>HTTP 200 Ok on success. HTTP 409 Conflict if provided an invalid store membership number.</returns>
      [HttpPost("validate")]
      public IActionResult ValidateMembershipNumber([FromBody] MembershipRequest request)
      {
         if (!IsStoreMembershipNumberValid(request.StoreMembershipNumber))
         {
            return GenerateErrorMessageWithMessage("Store membership number is not valid, it must be a multiple of 5!");
         }

         // Return the output claim(s)
         return Ok(new ValidationResponseContent
         {
            StoreMembershipNumber = request.StoreMembershipNumber.ToString()
         });
      }


      /// <summary>
      /// Constructs an HTTP 409 Conflict IActionResult to return back to the validation or orchestration step.
      /// This is used to communicate with the policy steps to communicate an error state.
      /// </summary>
      /// <param name="message">The message to be passed back to the user to explain why the request failed.</param>
      /// <returns>An IActionResult representing an HTTP 409 Conflict with a custom payload.</returns>
      private IActionResult GenerateErrorMessageWithMessage(string message)
      {
         return StatusCode((int)HttpStatusCode.Conflict, new StoreResponseContent
         {
            Version = "1.0.0",
            Status = (int) HttpStatusCode.Conflict,
            UserMessage = message
         });
      }


      /// <summary>
      /// Validates a provided store membership number by using the modulus operator (see more <see
          href="https://docs.microsoft.com/enus/dotnet/csharp/language-reference/operators/remainder-
operator">here</see>)
      /// to determine if the provided store membership number is a multiple of 5. If it is, we return true, if not, we
return
      /// false.
      /// </summary>
      /// <param name="storeMembershipNumber">The store membership number to be validated.</param>
      /// <returns>True on successful validation, false if validation fails.</returns>
      private bool IsStoreMembershipNumberValid(int storeMembershipNumber)
      {
         if (storeMembershipNumber % 5 != 0)
         {
            return false;
         }

         return true;
      }
   }
}
```

10. On the **File** menu, select **Save All**.
11. On the **Build** menu, select **Rebuild Solution**, and verify that the solution builds without any errors.

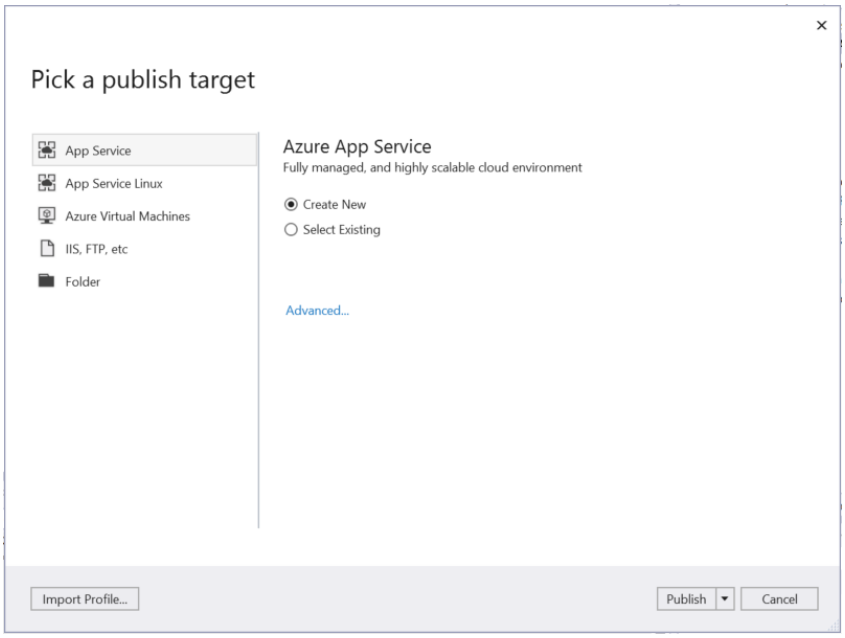
Publishing the project as an Azure Website

Proceed with the following steps:

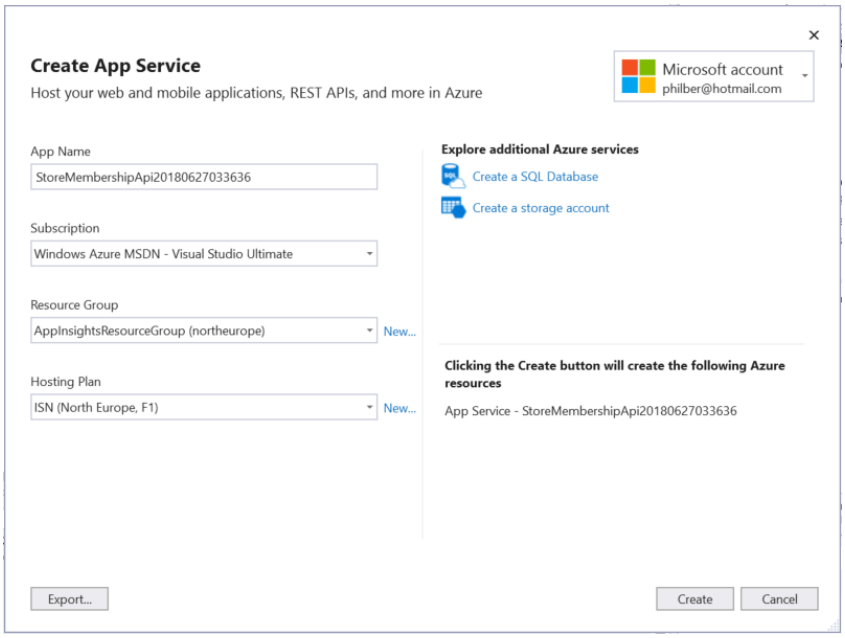
1. In Solution Explorer, right-click the **StoreMembershipApi** project node, and then select **Publish.**



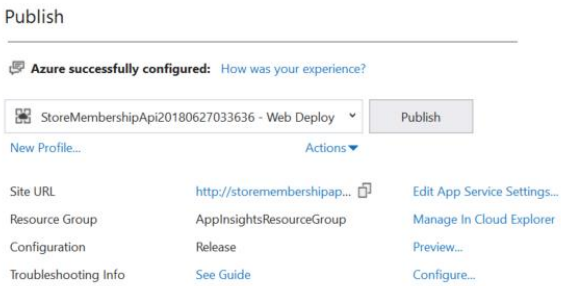
2. In the **Pick a publish target** dialog box, select **App Service**, select **Create New**, and then select **Publish**.



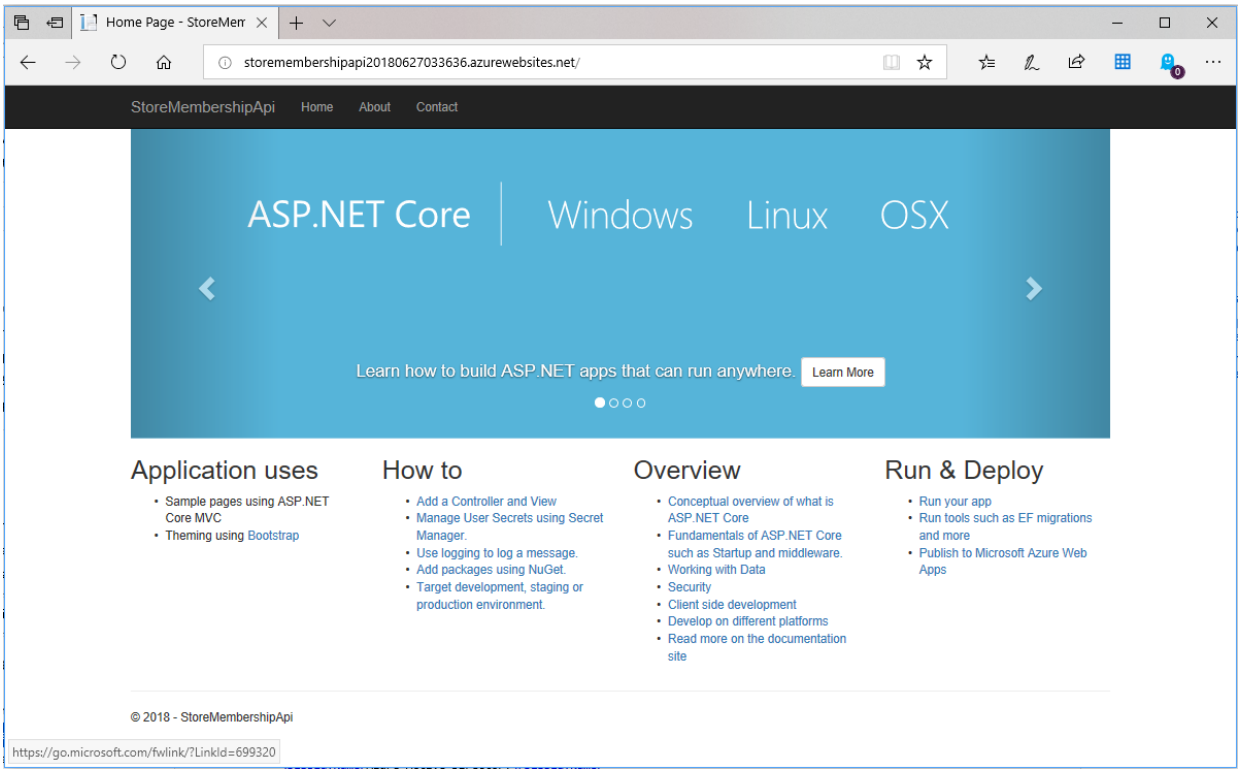
3. In the **Create App Service** dialog box, login to Azure as an account with administrative privileges in your Azure domain, specify a unique App Name, select a resource group, and then select **Create**.



4. Wait while the service is published.



5. The site will launch in the browser when publishing is complete.
6. Record the URL of your service in the browser address bar.



Modifying the API project to support user store membership date

Proceed with the following steps:

1. Return to the **StoreMembershipApi** project in Visual Studio 2017.
2. In Solution Explorer, right-click **Models**, select **Add**, and then select **Class.**
3. In the **Add New** Item dialog box, select **Visual C#**, and then select **Class**. Specify the name **MembershipDateResponseContent** and then select **Add**.
4. Replace the code in the *MembershipDateResponseContent.cs* file with the following code.

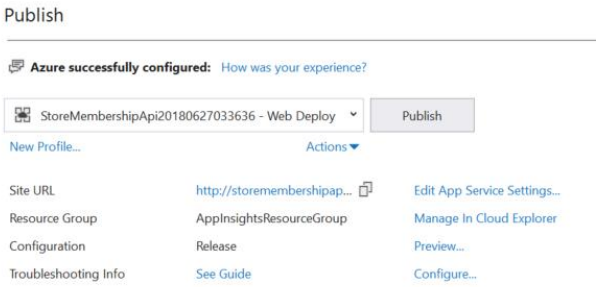
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace StoreMembershipApi.Models
{
   public class MembershipDateResponseContent : StoreResponseContent
   {
      public string StoreMembershipDate { get; set; }
   }
}
```

5. In Solution Explorer, expand the *Controllers* folder, and select the *MembershipController.cs* file.

6. Add the following methods to the **MembershipController** class.

```
/// <summary>
/// This method receives a storeMembershipnumber from the policy orchestration step and returns a 409 Conflict
/// response if the store membership number is not valid (not a multiple of 5), and a 200 Ok response containing
/// the member's membership date if the store membership number is valid.
/// </summary>
/// <param name="request">Passed from the policy orchestration step, contains a storeMembershipNumber.</param>
/// <returns>HTTP 200 Ok on success with an obtained membership date. HTTP 409
Conflict if provided an invalid store membership number.</returns>
[HttpPost("membershipdate")]
public IActionResult GetMembershipDate([FromBody] MembershipRequest request)
{
   if (!IsStoreMembershipNumberValid(request.StoreMembershipNumber))
   {
      return GenerateErrorMessageWithMessage("Store membership number is not valid, it must be a multiple of 5!");
   }

   var membershipDate = ObtainStoreMembershipDate();
   return Ok(new MembershipDateResponseContent
   {
       Version = "1.0.0",
       Status = (int)HttpStatusCode.OK,
       UserMessage = "Membership date located successfully.",
       StoreMembershipDate = membershipDate.ToString("d")
   });
}

/// <summary>
/// Generates a date within the last 90 days to return as the store membership number for a member.
/// In a production application you'd likely contact a database here to get a real membership date for a member.
/// </summary>
/// <returns>A DateTime representing the store membership date for a member.</returns>
private static DateTime ObtainStoreMembershipDate()
{
    var random = new Random();
    var daysOffOfToday = random.Next(0, 90);
    var randomDate = DateTime.Now.AddDays(-daysOffOfToday);

    return randomDate;
}
```

7. Save the file.
8. On the **Build** menu, select **Rebuild Solution**. Verify that the solution compiles without any errors.
9. In Solution Explorer, right-click the **StoreMembershipApi** project, and then select **Publish**.



10. In the **Publish** window, select **Publish**. Wait for the updated service to be deployed.

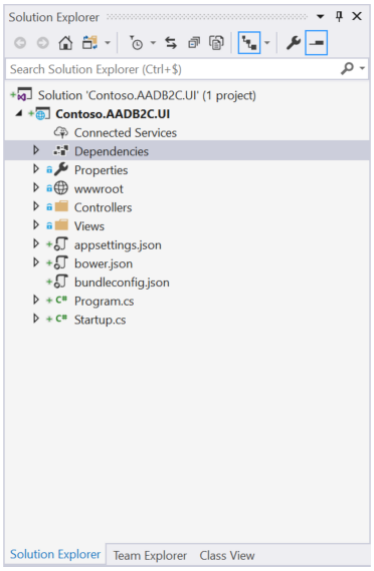
Building the Contoso.AADB2C.UI code sample

Building the code sample

Proceed with the following steps:

1. Navigate to the folder *scenarios\aadb2c-ief-ui-customization* of the "Starter Pack".
2. Extract on your computer the *Contoso.AADB2C.UI.zip* archive file under the **Starter-Pack** folder. A *Contoso.AADB2C.UI* folder is created underneath.
3. Using Visual Studio 2017, open the *Contoso.AADB2C.UI.sln* solution file in the *Contoso.AADB2C.UI* folder located under the **Starter-Pack** folder.

   If necessary, Visual Studio 2017 will download all the required NuGet package and resolve all the dependencies.



4. Open the *Controllers\HomeController.cs* file and scroll to the **unified** method.

```
public IActionResult unified(string campaignId)
{
   SetPageBackground(campaignId);
   return View();
}
```

   This method accepts a **campaignId** parameter.

5. Navigate to the definition of **SetPageBackground** method.

```
private void SetPageBackground(string campaignId)
{
   // If campaign ID is Hawaii, show Hawaii backgound
   if (campaignId != null && campaignId.ToLower() == "hawaii")
   {
      ViewData["background"] = "https://kbdevstorage1.blob.core.windows.net/asset-blobs/19889_en_1";
   }
   // If campaign ID is Tokyo, show Tokyo backgound
   else if (campaignId != null && campaignId.ToLower() == "tokyo")
   {
      ViewData["background"] = "https://kbdevstorage1.blob.core.windows.net/asset-blobs/19666_en_1";
   }
   // Default background
   else
   {
      ViewData["background"] = "https://kbdevstorage1.blob.core.windows.net/asset-blobs/18983_en_1";
   }
}
```

This method checks the parameter's value and sets the ViewData["background"] variable accordingly to replace the background image of the view:

- https://kbdevstorage1.blob.core.windows.net/asset-blobs/19889_en_1

-or-

- https://kbdevstorage1.blob.core.windows.net/asset-blobs/19666_en_1

-or-

- https://kbdevstorage1.blob.core.windows.net/asset-blobs/18983_en_1

6. Open the *Views\Home\unified.cshtml* file and scroll to the **unified** method and locate the element under the <body> element.

```
<body>
   <div id="background_branding_container">
      <img id="background_background_image" src="@ViewData["background"]" alt="Illustration" />
   </div>
     …
</body>
```

The <img> element with id background_background_image is set with the above picture files depending on the value of the **campaignId** parameter.

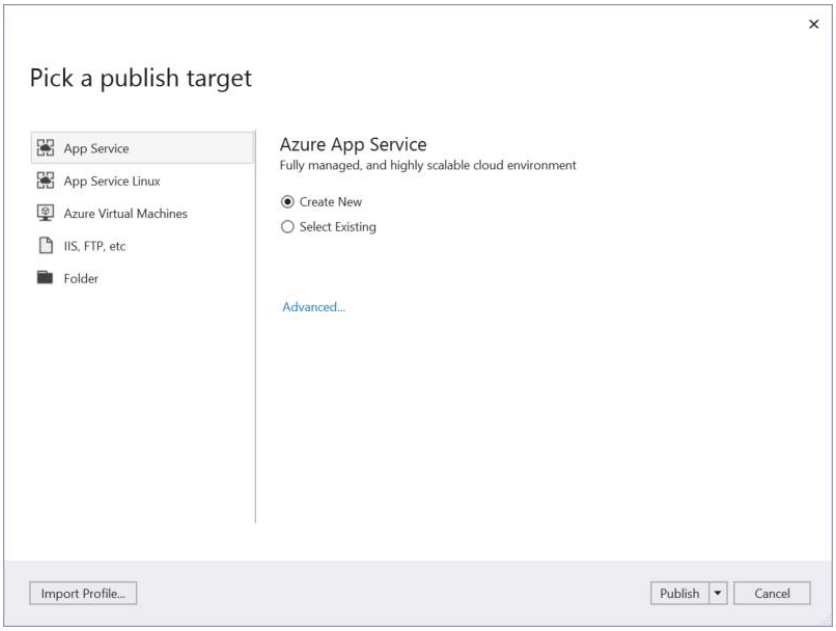
The **selfAsserted** method in the *Controllers\HomeController.cs* file and the <img> element with id background_background_image in the *Views\Home\unified.cshtml* file are set the same way.

7. On the **Build** menu, select **Rebuild Solution**. Verify that the solution compiles without any errors.

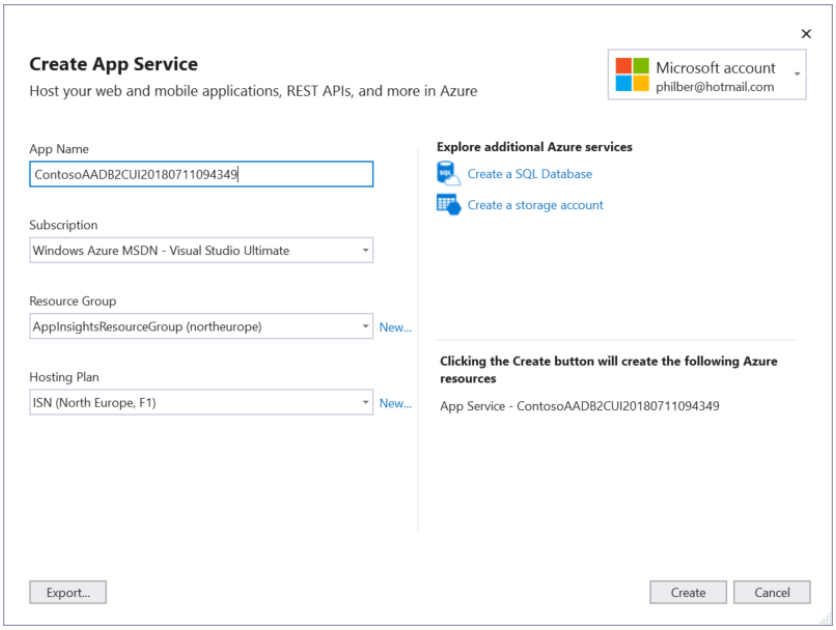
Publishing the code sample to Azure

Proceed with the following steps:

1. In Solution Explorer, right-click the **Contoso.AADB2C.UI** project, and then select **Publish**. A **Pick a publish target** dialog opens.



2. Select the **App Service**, select **Create New**, and then click **Publish**. A **Create App Service** dialog opens.

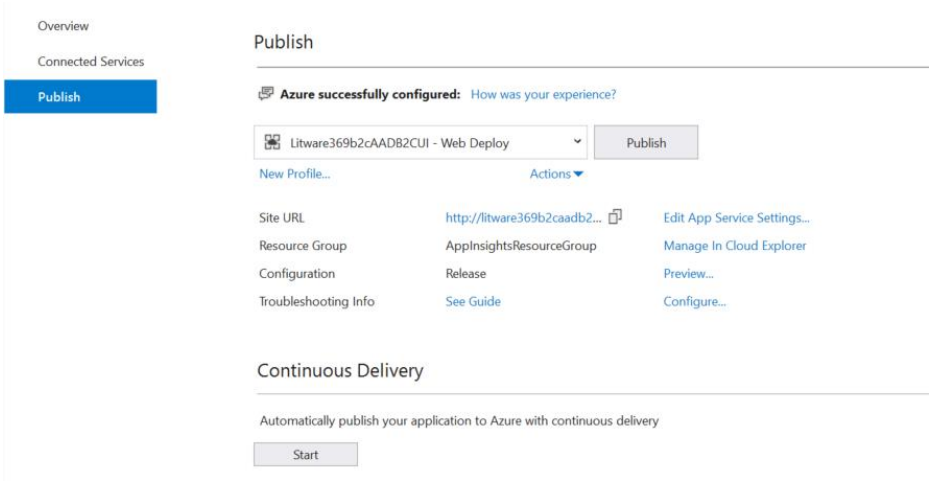


This dialog enables you to create all the necessary Azure resources needed to run the ASP.NET Core web app in Azure.

3. In **App Name**, accept the automatically generated name, which is unique, or type a unique app name - valid characters are a-z, A-Z, 0-9, and the hyphen (-) -. For example, in our configuration, **Litware369b2cAADB2CUI**.

   The URL of the web app will be http://<*your_app_name*>.azurewebsites.net, where <*your_app_name*> is the chosen app name for the web app.

4. Select **Create** to start creating the Azure resources. The deployment starts.

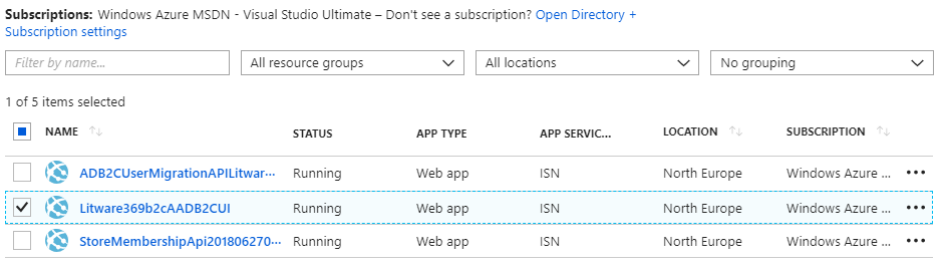


5. After the creation process is complete, the wizard publishes the ASP.NET web app to Azure and then launches the app in the default browser.

Configuring CORS in Azure App Service

Proceed with the following steps:

1. In the Azure portal, Select **App Services**, and then select the name of the web app. For example, in our configuration, **Litware369b2cAADB2CUI**.



2. Under **API**, select **CORS**.

3. In **ALLOWED ORIGINS**, do either of the following:

   - Enter the URL or URLs that you want to allow JavaScript calls to come from

   -or-

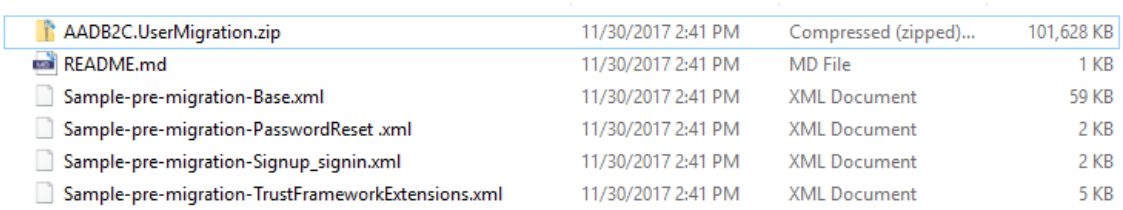
   - Enter an asterisk (*) to specify that all origin domains are accepted.

4. Click **Save**. After you select **Save**, the API app accepts JavaScript calls from the specified URLs.

Building the AADB2C.UserMigration code sample

This section depicts how to build the AADB2C.UserMigration sample.

The folder *scenarios\aadb2c-user-migration* of the "Starter Pack" provide both a code sample application and a pre-defined set of custom policies to ease such a migration.



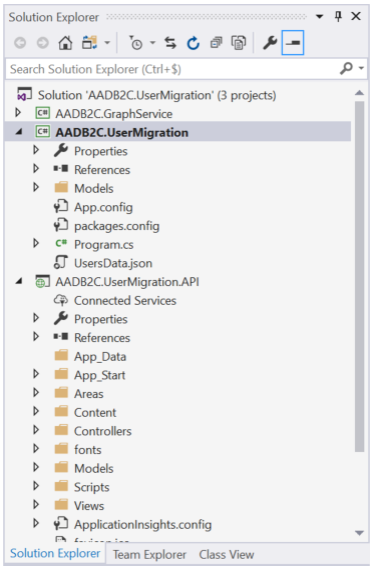
This AADB2C.UserMigration code sample along with the provided XML files demonstrate how to migrate existing user accounts, from any identity provider to your B2C tenant. This code sample is not meant to be prescriptive, but rather describes two of several different approaches as outlined in the course of this paper (see section § *Understanding the primary considerations for the migration*). The developer is responsible for suitability and performances.

Building and running the AADB2C.UserMigration project

Proceed with the following steps:

1. Extract on your computer the *AADB2C.UserMigration.zip* archive file under the **Starter-Pack** folder. An *AADB2C.UserMigration* folder is created underneath.
2. Using Visual Studio 2017, open the *AADB2C.UserMigration.sln* solution file in the *AADB2C.UserMigration* folder located under the **Starter-Pack** folder.

   If necessary, Visual Studio 2017 will download all the required NuGet package and resolve all the dependencies.



3. In Solution Explorer, under the **AADB2C.UserMigration** project, open the *App.config* file.

4. In the *App.config* file, modify the values in the **appSettings** section. Provide the name of your B2C tenant, the application ID and the secret key for the application.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6.1" />
  </startup>

  <appSettings>

    <add key="b2c:Tenant" value="" />
    <add key="b2c:ClientId" value="" />
    <add key="b2c:ClientSecret" value="" />

    …
  </appSettings>
…
</configuration>
```

You can reuse here the application ID and secret that you created for the **B2CGraphClient** application in the third document of this series, see section § *Building the B2CGraphClient code sample* in Appendix B. Building the code samples:

a. Tenant: litware369b2c.onmicrosoft.com
b. ClientId: eca22455-926d-43db-89de-1cfafe9a05e6
c. ClientSecret: /NC2fqSQjahypbcYikFh6ebYplPls5f+ZsxOlm2qzCc=

5. On the **Build** menu, select **Rebuild Solution**. Verify that the solution compiles without any errors.
6. In Solution Explorer, under the **AADB2C.UserMigration** project, select the file *Program.cs*. The **Main** method in this file is the entry point for the application.

   The application expects the user to provide a numeric option on the command line to specify which operation to perform. If no option is specified, the code displays a menu indicating the available choices - there are other selections available other than 1 and 2, but they are not used by this illustration -.

   If the user specifies option 1, the application invokes the **MigrateUsersWithPasswordAsync** method.

   If the user invokes option 2, the application runs the **MigrateUsersWithRandomPasswordAsync** method.

```
static void Main(string[] args)
{
   if (args.Length <= 0)
   {
      Console.WriteLine("Please enter a command as the first argument.");
      Console.WriteLine("\t1                  : Migrate users with password");
      Console.WriteLine("\t2                  : Migrate users with random password");
      Console.WriteLine("\t3 Email-address  : Get user by email address");
      Console.WriteLine("\t4 Display-name   : Get user by display name");
      Console.WriteLine("\t5                : User migration cleanup");
      return;
    }
    try
    {
       switch (args[0])
       {
          case "1":
             MigrateUsersWithPasswordAsync().Wait();
             break;
          case "2":
             MigrateUsersWithRandomPasswordAsync().Wait();
```

```
            break;
            …
        }
        …
    }
    …
}
```

7. Scroll down and find the **MigrateUsersWithPasswordAsync** method. This method reads the user data from a file (checking to make sure that the file exists first) and creates a collection of user objects in a variable called **users**. The code then iterates through this collection and calls the **b2CGraphClient.CreateUser** method to create each user. Note that the **users.GenerateRandomPassword** variable is a Boolean indicating whether the B2CGraphClient.CreateUser method should use the password specified as a parameter to this method or generate its own random password. This method sets the **GenerateRandomPassword** variable to **false**.

Note The method **MigrateUsersWithRandomPasswordAsync**, which runs when the users specify option 2 when executing the program, sets this variable to **true**. The **b2cGraphClient.CreateUser** method uses the Azure AD Graph API to create the user by sending an HTTP POST request to the **users** resource for Azure AD, as previously described.

```
/// <summary>
/// Migrate users with their password
/// </summary>
/// <returns></returns>
static async Task MigrateUsersWithPasswordAsync() {
    string appDirectoryPath = Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Location);
    string dataFilePath = Path.Combine(appDirectoryPath, Program.MigrationFile);

    // Check file existence
    if (!File.Exists(dataFilePath))
    {
       Console.ForegroundColor = ConsoleColor.Red;
       Console.WriteLine($"File '{dataFilePath}' not found");
       Console.ResetColor();
       return;
    }

    // Read the data file and convert to object
    UsersModel users = UsersModel.Parse(File.ReadAllText(dataFilePath));

    // Create B2C graph client object
    B2CGraphClient b2CGraphClient = new B2CGraphClient(Program.Tenant, Program.ClientId, Program.ClientSecret);
    foreach (var item in users.Users)
    {
        await b2CGraphClient.CreateUser(item.email, item.password, item.displayName, item.firstName, item.lastName,
                                        users.GenerateRandomPassword);
    }

    Console.WriteLine("Users migrated successfully");
}
```

Building and running the AADB2C.UserMigration.API project

To build and deploy the AADB2C.UserMigration.API, you will have to:

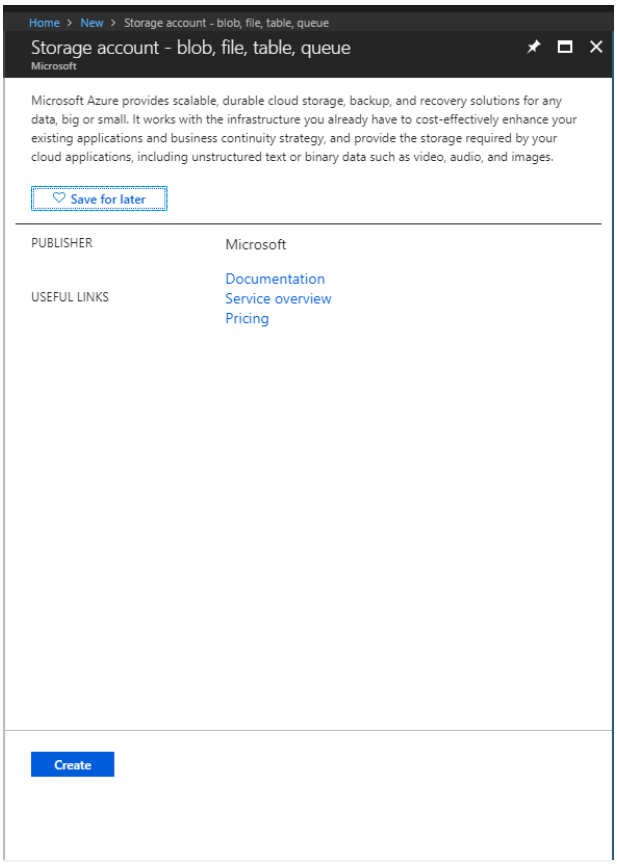
1. Create a storage account.
2. Building and deploying the AADB2C.UserMigration.API project.
3. Updating the “Stater Pack” custom policies.

The next section depicts in order each of these related tasks.

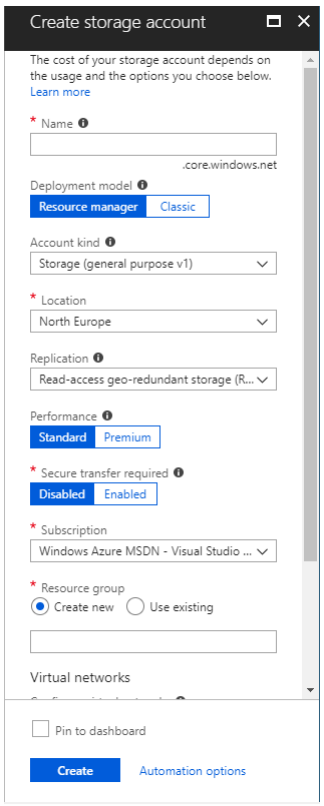
Creating a storage account

The following procedure shows how to implement this approach.

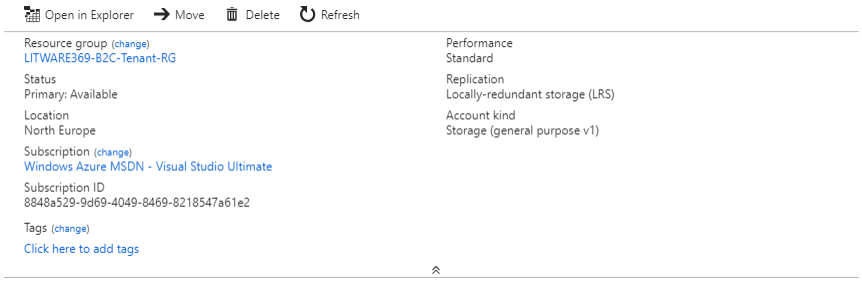
1. In the Azure portal, make sure you are connected as an account with administrative privileges, and switch to your Azure tenant (not the B2C tenant).
2. Select **+ Create a resource**.
3. In **Search**, enter "*Storage account*", and then select **Storage account – blob, table, file, queue**.



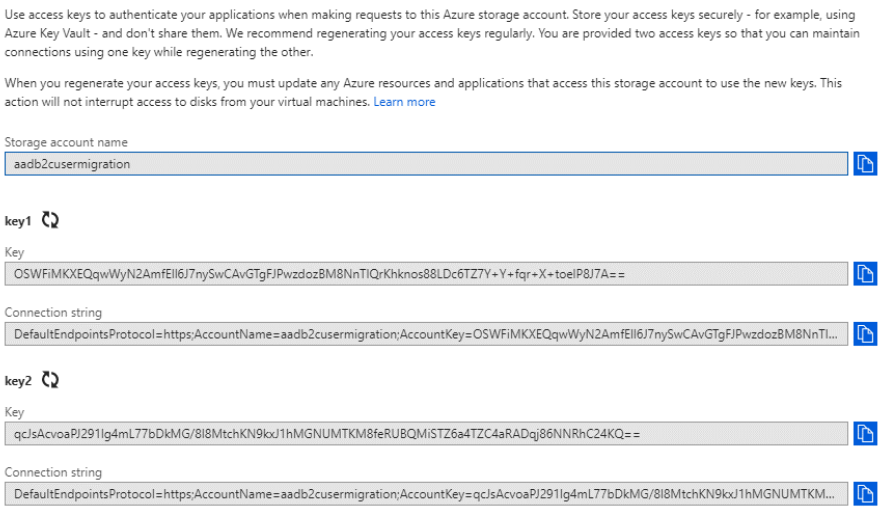
4. Click **Create**.



a. Enter a unique name for the storage account, for example in our illustration "*aadb2cusermigration*".
b. Leave the remaining options at their defaults (create a new resource group if necessary), and then select **Create**.



5. When the storage account has been created, select **All resources**, select your new storage account, and then select **Access keys** under **SETTINGS**.



6. Under **key1**, make of note the value for the **Connection string**:

   DefaultEndpointsProtocol=https;AccountName=aadb2cusermigration;AccountKey=OSWFiMKXEQqwWyN2AmfEll6J7nySwCAvGTgFJPwzdozBM8NnTlQrKhknos88LDc6TZ7Y+Y+fqr+X+toelP8J7A==;EndpointSuffix=core.windows.net.

Building and deploying the AADB2C.UserMigration.API project

Proceed with the following steps:

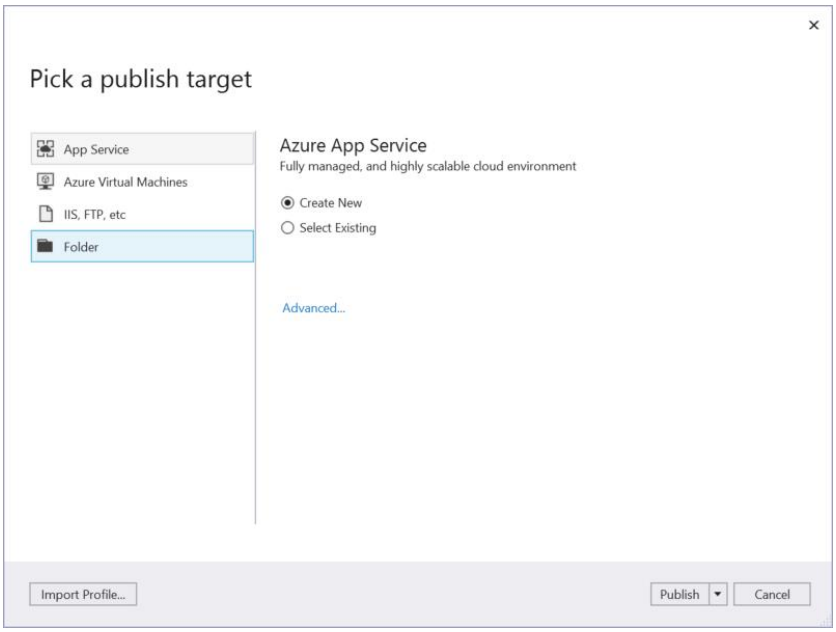
1. In Visual Studio, return to the *AADB2C.UserMigration.sln* solution file in the *AADB2C.UserMigration* folder located under the **Starter-Pack** folder.
2. In Solution Explorer, expand the **AADB2C.UserMigration** project, and then select the *App.config* file.
3. In the *App.config* file, in the **appSettings** section, insert the above table storage connection string as the value for the **BlobStorageConnectionString** key.

```
<appSettings>
   <add key="b2c:Tenant" value="litware369b2c.onmicrosoft.com" />
   <add key="b2c:ClientId" value="eca22455-926d-43db-89de-1cfafe9a05e6" />
   <add key="b2c:ClientSecret" value="/NC2fqSQjahypbcYikFh6ebYplPls5f+ZsxOlm2qzCc=" />
   <add key="MigrationFile" value="UsersData.json" />

   <add key="BlobStorageConnectionString" value="" />

</appSettings> </appSettings>
```

4. In Solution Explorer, expand the **AADB2C.UserMigration.API** project, and then select the *Web.config* file.
5. In the *Web.config* file, in the **appSettings** section, insert the same table storage connection string as the value for the **BlobStorageConnectionString** key.

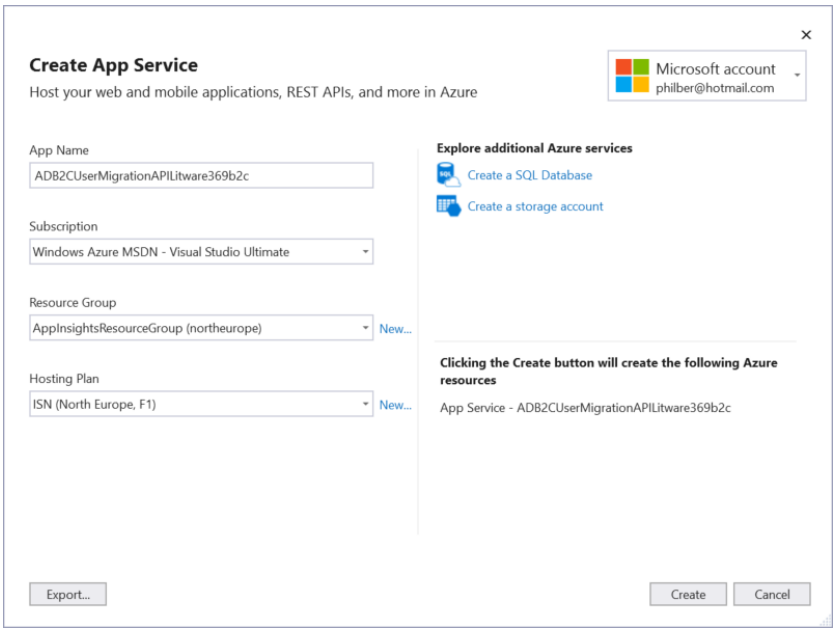
```
<appSettings>
   <add key="webpages:Version" value="3.0.0.0" />
   <add key="webpages:Enabled" value="false" />
   <add key="ClientValidationEnabled" value="true" />
   <add key="UnobtrusiveJavaScriptEnabled" value="true" />

   <add key="BlobStorageConnectionString" value="" />

</appSettings>
```

6. On the **Build** menu, select **Rebuild Solution**.
7. In Solution Explorer, right-click the **AADB2C.UserMigration.API** project, and then select **Publish**. A dialog box pops up.



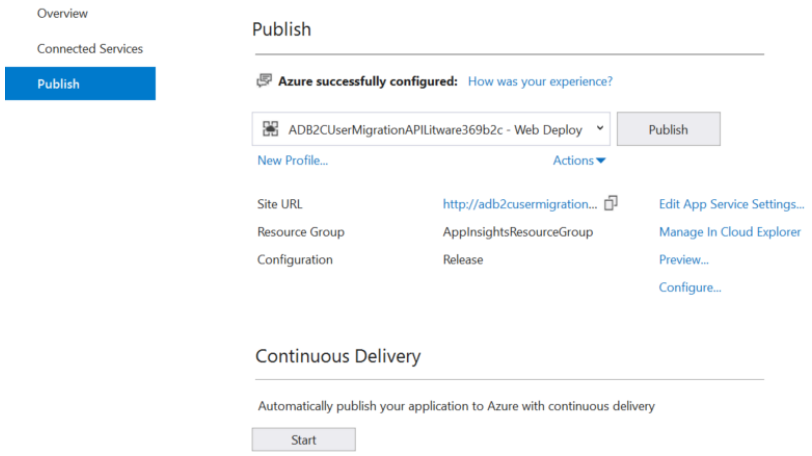
8. In the **Pick a publish target** dialog, select **App Service**, select **Create New**, and then select **Publish**.
9. In the **Create App Service** dialog box:
    a. Log in with an account with administrative privileges in your Azure tenant (if necessary, add the account to Visual Studio).

b. Set the **App Name** to **ADB2CUserMigrationAPILitware369b2c** where **Litware369b2c** is the name of your B2C tenant.
c. Accept the default values for the remaining fields.

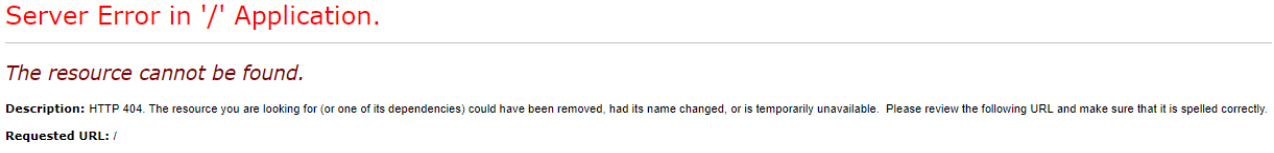


d. And then select **Create**.

9. Wait while the web application is published.



When deployment is complete, the web app will start running and a browser window will open. The browser might display the error message "The resource cannot be found", but this is OK ;-)



Updating the AADB2C.UserMigration project

Proceed with the following steps:

1. In Visual Studio, return to the *AADB2C.UserMigration.sln* solution file in the *AADB2C.UserMigration* folder located under the **Starter-Pack** folder.

10. In Solution Explorer, expand the **AADB2C.UserMigration** project, and then open the *App.config* file.

11. In the *App.config* file, in the **appSettings** section of the file, modify the value is the **MigrationFile** key to refer to a file named *UsersDataResetPasswords.json*.

```
<appSettings>
   <add key="b2c:Tenant" value="litware369b2c.onmicrosoft.com" />
   <add key="b2c:ClientId" value="eca22455-926d-43db-89de-1cfafe9a05e6" />
   <add key="b2c:ClientSecret" value="/NC2fqSQjahypbcYikFh6ebYplPls5f+ZsxOlm2qzCc=" />

   <add key="MigrationFile" value="UsersDataResetPasswords.json " />

   <add key="BlobStorageConnectionString" value="" />
</appSettings>
```

12. In Solution Explorer, in the **AADB2C.UserMigration** project, open the *Program.cs* file.

13. Scroll down and find the **MigrateUsersWithRandomPasswordAsync** method.

```
static async Task MigrateUsersWithRandomPasswordAsync()
{
   string appDirecotyPath = Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Location);
   string dataFilePath = Path.Combine(appDirecotyPath, Program.MigrationFile);

   // Check file existence
   if (!File.Exists(dataFilePath))
   {
      Console.ForegroundColor = ConsoleColor.Red;
      Console.WriteLine($"File '{dataFilePath}' not found");
      Console.ResetColor();
      return;
    }

    // Read the data file and convert to object
    UsersModel users = UsersModel.Parse(File.ReadAllText(dataFilePath));

    // Create B2C graph client object
    B2CGraphClient b2CGraphClient = new B2CGraphClient(Program.Tenant, Program.ClientId, Program.ClientSecret);

    // Parse the connection string and return a reference to the storage account.
    CloudStorageAccount storageAccount = CloudStorageAccount.Parse(Program.BlobStorageConnectionString);

    // Create the table client.
    CloudTableClient tableClient = storageAccount.CreateCloudTableClient();

    // Retrieve a reference to the table.
    CloudTable table = tableClient.GetTableReference("users");

    // Create the table if it doesn't exist.
    table.CreateIfNotExists();

    // Create the batch operation.
    TableBatchOperation batchOperation = new TableBatchOperation();
    foreach (var item in users.Users)
    {
       await b2CGraphClient.CreateUser(item.email, item.password, item.displayName, item.firstName, item.lastName,
                                       users.GenerateRandomPassword);

        // Create a new customer entity.
        // Note: Azure Blob Table query is case sensitive, always set the email to lower case
        TableEntity user = new TableEntity("B2CMigration", item.email.ToLower());

        // Create the TableOperation object that inserts the customer entity.
```

```
        TableOperation insertOperation = TableOperation.InsertOrReplace(user);

        // Execute the insert operation.
        table.Execute(insertOperation);
    }

    Console.WriteLine("Users migrated successfully");
}
```

The above code uses the Azure Table API to create a new table, named **users**, in your storage account. It then iterates through the user listed in the JSON file and called the

b2cGraphClient.CreateUser method to add each user to your Azure AD domain. If this operation is successful, the method then adds a record of the user to the **users** table.

14. In the statement that calls the **b2cGraphClient.CreateUser** method, change the **users.GenerateRandomPassword** parameter (highlighted) to **false**. For this example, we want to preserve user's passwords, but force them to change in the next time they log in.

```
await b2CGraphClient.CreateUser(item.email, item.password, item.displayName, item.firstName, item.lastName,  false);
```

15. On the **Build** menu, select **Rebuild Solution**.